Titre: Title:	Machine décisionnelle pour systèmes multi-robots coopératifs
Auteur: Author:	Julien Beaudry
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Beaudry, J. (2005). Machine décisionnelle pour systèmes multi-robots coopératifs [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7588/

UNIVERSITÉ DE MONTRÉAL

MACHINE DÉCISIONNELLE POUR SYSTÈMES MULTI-ROBOTS COOPÉRATIFS

JULIEN BEAUDRY
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
DÉCEMBRE 2005

Your file Votre référence
ISBN: 978-0-494-16754-0
Our file Notre référence
ISBN: 978-0-494-16754-0

UNIVERSITÉ DE MONTRÉAL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

MACHINE DÉCISIONNELLE POUR SYSTÈMES MULTI-ROBOTS COOPÉRATIFS

présenté par : BEAUDRY Julien
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. BRAULT Jean-Jules, Ph.D., président
M. HURTEAU Richard, D.Ing., membre et directeur de recherche
M. GOURDEAU Richard, Ph.D., membre et codirecteur de recherche
M. DAGENAIS Michel, Ph.D., membre

REMERCIEMENTS

Je tiens tout d'abord à remercier Richard Hurteau, mon directeur de recherche, pour son support moral et scientifique mais aussi pour toutes les ressources matérielles et humaines qu'il a été en mesure de joindre au projet de robots joueurs de soccer de l'École Polytechnique. C'est en grande partie grâce à cet apport que le présent projet de maîtrise a été possible. De plus, Richard Gourdeau a offert à ce projet un support scientifique très apprécié. En particulier, les apports de M.Gourdeau et M.Hurteau ont permis à un autre étudiant, Sylvain Marleau, de compléter un projet de maîtrise en lien avec le projet présenté dans ce mémoire. Le projet de M.Marleau ayant augmenté considérablement les capacités de perception et de fonctionnement autonome des robots, je tiens également à remercier ce dernier pour son travail exemplaire. Le projet de robots joueurs de soccer a un historique de développement particulier, étant le résultat de travaux pédagogiques, scientifiques et aussi émanant de travaux de sociétés techniques. J'aimerais donc remercier toutes les personnes qui ont été impliquées dans les nombreuses étapes de développement : les membres de *SAE Robotique* de l'époque, les auteurs de la librairie *MICROB* à l'IREQ, Romano M. DeSantis, Richard Grenier, Félix Duchesneau, Alexandre Forget, Cédric Poirier et bien sûr les membres actuels du projet *Robofoot*.

Je tiens à remercier quelques personnes bien particulières pour leur support moral inconditionnel. Mes parents ont toujours été là pour me supporter et m'encourager à donner le meilleur de moi-même. Je leur dois énormément. Mon dernier hommage et remerciement est dédié à ma conjointe, Stéphanie, qui a dû accepter le fait que je demeure hypnotisé durant d'innombrables heures devant un écran d'ordinateur. Pendant ce temps elle a su prendre soins de façon admirable de deux rayons de soleil, Béatrice et Charlotte.

Ce projet a pu être réalisé grâce au support financier du Fonds québécois de la recherche sur la nature et les technologies ainsi que celui du Département de génie électrique de l'École Polytechnique de Montréal.

RÉSUMÉ

Dans ce mémoire, on vise à développer une nouvelle architecture décisionnelle capable d'adresser la problématique des systèmes multi-robots coopératifs. L'architecture développée se veut une architecture hybride, c'est-à-dire une architecture dans laquelle on retrouve des capacités réactives, soient des comportements et du contrôle de bas niveau, mais également des capacités délibératives, soient des mécanismes décisionnels capables de niveaux d'abstraction et de prédiction sur un horizon de temps qui sont arbitraires et variables.

L'architecture développée a été nommée machine décisionnelle hiérarchique puisqu'elle utilise une structure hiérarchique à représentation très simple où le fonctionnement se résume à une succession séquentielle de mécanismes décisionnels. Une ligne décisionnelle se termine avec un certain comportement dans le but d'activer les actions offertes par le système robotisé. La machine décisionnelle hiérarchique est conçue dans le but d'être distribuée au sein de chacun des robots d'un système multi-robots. La prise de décision se fait donc de façon distribuée et afin d'instaurer une coopération entre les robots d'un système, par exemple pour le partage de ressources ou l'attribution dynamique de rôles, on peut utiliser des règles préétablies intégrées au sein des machines distribuées. Il peut cependant arriver qu'un système à prise de décision distribuée soit la source de conflits décisionnels malgré les règles de coopération préétablies, en particulier quand la perception l'est également. Dans ce cas, au lieu de privilégier une négociation entre robots, le concept de supervision décisionnelle est utilisé. Quand cela est jugé nécessaire, un processus centralisé utilisant une approche clients/serveurs peut se charger de superviser la prise de décision distribuée et de détecter les situations qui demandent intervention. La supervision décisionnelle est possible sur toute la hiérarchie d'une machine décisionnelle donnée.

Afin de tester et valider l'architecture décisionnelle développée, une équipe de robots joueurs de soccer développée à l'École Polytechnique a servi de banc d'essai. Les robots joueurs de soccer permettent de mettre en pratique la machine décisionnelle hiérarchique ainsi que la supervision décisionnelle. Le projet *Robofoot* constitue le seul système multi-robots en Amérique du Nord à participer à la « Middle Size Robot League » de la *RoboCup*, la ligue qui se rapproche le plus d'un jeu réaliste tel que joué par l'être humain. Au cours de l'année 2005, l'équipe de robots

joueurs de soccer a participé à deux compétitions internationales fédérées par la communauté scientifique de la *RoboCup*. L'aspect quantitatif et compétitif du soccer robotisé permet d'évaluer directement la performance des solutions développées. Par les compétitions et les nombreux essais effectués, il a été possible de prendre des mesures de performance. Les divers résultats obtenus indiquent des performances et caractéristiques très intéressantes répondant aux besoins spécifiques des systèmes multi-robots autonomes à prise de décision temps réel et distribuée.

Mots-clés :
système multi-robots coopératif, architecture de contrôle, supervision décisionnelle, soccer robotisé.

ABSTRACT

In this thesis, it is intended to develop a novel decision architecture aimed at addressing the problematic of cooperative multi-robot systems. The architecture resulting is considered hybrid, in other words an architecture in which reactive capabilities are found working in conjunction with deliberative capabilities which are decision mechanisms dealing with various levels of abstraction and prediction on an arbitrary time horizon.

The architecture developed is called Hierarchical Decision Machine since it uses a hierarchical structure really simply represented and it works as a succession of sequential decision taking mechanisms. A decision line is terminated with a behaviour willing to activate various actions offered by the robotic system. The Hierarchical Decision Machine is designed with the objective of being distributed on every robot of a multi-robot system. Decision taking mechanisms are then implemented in a distributed fashion and in order to introduce cooperative behaviours between robots of a system, for example resources sharing or dynamic role allocation, pre-established agreements can be used on each decision machine. It is however possible that a certain system, in particular if perception capabilities are also distributed, is the source of decision conflicts between robots. In this case, instead of using negotiation between robots, the concept of Decision Supervision is introduced. When it is judged necessary, a centralized process using a client/server approach can handle the supervision of distributed decision taking and detect situations where intervention is appropriate. Decision supervision is possible on the complete hierarchy of a given decision machine.

In order to test and validate the resulting decision architecture, a team of soccer-playing robots developed at École Polytechnique is used as a test bench. The soccer-playing robots allow putting in practice the Hierarchical Decision Machine completed with Decision Supervision. The *Robofoot* project represents the only North American multi-robot system aimed at competing in the Middle Size Robot League of the *RoboCup* Federation. This league is the one offering the most realistic game as played by humans. During the year 2005, the team of robots participated in two international competitions organized under the scientific supervision of the *RoboCup* community. The quantitative possibilities and competitive aspect of robotic soccer make it a great

test bench to evaluate direct performance of developed solutions. With the competitions and numerous experimentations achieved, it has been possible to take many performance measures concerning the concept developed in this thesis. The results obtain are indicate much interesting performance as well as characteristics to address the specific needs of autonomous multi-robot systems with real-time distributed decision taking.

Keywords :

cooperative multi-robot system, control architecture, decision supervision, robotic soccer.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

INTRODUCTION

Le texte qui suit vise à présenter différents éléments novateurs développés à l'École Polytechnique de Montréal dans le cadre d'un projet de système multi-robots autonome et coopératif. Afin de compléter les fonctionnalités du système déjà développées, il était nécessaire de s'attarder à la question de l'architecture de contrôle nécessaire à la gestion d'une équipe de robots autonome. Plus spécifiquement, la question de l'architecture décisionnelle se devait d'être approfondie. Les concepts utilisés pour ce qui traite de l'architecture de contrôle dans son ensemble étaient dès le départ assez bien définis dans le cadre du projet. Cependant, la question de la prise de décision, se voulant distribuée au sein d'une équipe de robots, restait ouverte. Ainsi, le présent projet de maîtrise traite du développement d'une architecture décisionnelle pour systèmes multi-robots coopératifs, laquelle doit s'intégrer dans un ensemble plus complet qui est l'architecture de contrôle du système, cette dernière architecture pouvant varier en fonction de l'application visée. Afin de valider les développements conceptuels effectués, l'architecture décisionnelle proposée fut mise en pratique grâce à une programmation adéquate. Cette implantation logicielle fut testée dans le système multi-robots déjà développé, soit les robots joueurs de soccer (Robofoot ÉPM, [36]). En plus de développer certains mécanismes décisionnels appropriés pour ce système et d'effectuer plusieurs essais pour tester et valider le tout, le projet a été placé en situation de compétition, face à d'autres projets semblables, lors de tournois internationaux de soccer robotisé.

Les pages qui suivent présentent donc le contexte de développement du projet, les divers éléments développés, certains résultats obtenus ainsi que leur analyse et une discussion sur la pertinence des développements.

0.1 Motivation pour les systèmes multi-robots coopératifs

Pourquoi l'être humain a-t-il cherché à vivre en société et à développer des structures et conventions sociales aussi élaborées ? C'est peut-être inconsciemment et grâce au fil du temps que cet aspect social s'est autant développé, mais le résultat, toujours en évolution, est impressionnant. La compréhension du monde et l'utilisation qu'on en fait progresse constamment

grâce au savoir commun. Les exemples concrets de coopération entre humains et d'interaction sociale sont innombrables.

Pour la robotique, le fait d'utiliser une équipe de robots, de leur inculquer certains concepts sociaux, et non seulement d'utiliser un seul robot individuel peut avoir un impact majeur sur les capacités offertes par les systèmes robotisés. Il n'y aura peut-être jamais de groupes de robots suffisamment évolués pour que l'on puisse utiliser le terme de « société de robots », mais le regroupement de robots au sein d'un même système est aujourd'hui possible et d'un intérêt majeur. On peut certainement comparer une équipe de robots à certains regroupements d'individus ayant un but fonctionnel commun, par exemple des membres d'une équipe sportive ou d'une équipe de travail étant en mesure de s'entraider et de coopérer. Plus spécifiquement, la coopération au sein de systèmes multi-robot devrait être en mesure d'apporter les avantages suivants :

- Tolérance aux fautes : un groupe de robots peut continuer de fonctionner même si des fautes se produisent au sein d'un ou de plusieurs robots du groupe. Par exemple, un robot en panne peut être remplacé dans l'exécution de ses tâches.
- Meilleure efficacité et/ou performance : un groupe de robots peut offrir un gain en efficacité et également en performance (temps d'exécution réduit, retour sur investissement, etc.).
- Extension et/ou perfectionnement de l'ensemble de tâches : un groupe de robots peut être en mesure d'accomplir de nouvelles tâches ou encore de perfectionner les tâches déjà accomplies.
- Réduction de coût : quoique ce point n'étant pas toujours véridique, il peut-être possible de développer un groupe de robots relativement simples à moindre coût comparativement à un seul robot beaucoup plus complexe, dans le but d'effectuer le même ensemble de tâches.

Il est possible donc de croire que dans presque tous les domaines où la robotique mobile peut être utile, les systèmes multi-robots peuvent apporter des nouvelles possibilités intéressantes par rapport aux solutions plus conventionnelles à un seul robot ou encore à plusieurs robots mais sans notion de système intégrant l'ensemble de ces robots. Parmi les exemples d'applications de tels systèmes, nous pouvons penser aux suivants : équipe de robots pour exploration spatiale,

équipe de robots pour une usine de fabrication ou entrepôt de distribution, équipe de robots pour détecter et enlever les mines anti-personnel, équipe de robots de sécurité, équipe de robots pour inspection d'endroits à risque (grotte, mines, sous-marin, aérien). Des possibilités d'applications sont également à prévoir dans d'autres secteurs comme l'industrie agricole et celle des transports.

0.2 Utilisation du soccer robotisé comme plateforme d'essai

Le soccer robotisé constitue la plateforme d'essai préconisée par le présent projet. Pourquoi donc utiliser le soccer robotisé comme prétexte de développement d'un système multi-robots? En fait, la problématique de développer une équipe de robots joueurs de soccer autonomes demande le développement de systèmes complexes dans des domaines variés (systèmes électromécaniques, informatique temps-réel, systèmes de perception, contrôle de bas et de haut niveau, mécanismes de coopération, intelligence artificielle, etc.). Comme la plupart des projets de robotique mobile demandent la résolution de problèmes similaires (mouvements agiles, perception de l'environnement, etc.), les contributions provenant des développements reliés au soccer robotisé sont considérables. En particulier, l'étude des systèmes multi-robots coopératifs y est un sujet prédominant.

Comme le soccer est le sport le plus populaire au monde, les compétitions sont une bonne occasion de regrouper la communauté scientifique internationale provenant des divers domaines rattachés. De plus, les matchs joués entre deux équipes offrent un environnement compétitif ainsi qu'un aspect quantitatif qui sont très utiles à l'évaluation de performance. Il existe d'ailleurs de nombreuses compétitions de soccer robotisé à travers le monde, comme la Coupe du Monde de soccer robotisé, la *RoboCup*.

La *RoboCup* est une compétition internationale permettant à des équipes de robots de différentes universités et même d'entreprises privées de s'affronter dans un tournoi qui a lieu annuellement dans divers coins de la planète. Plusieurs ligues permettent de spécifier des formats de terrain et de robots différents, dont la « Middle-Size Robot League », la ligue qui se rapproche le plus actuellement du soccer joué par les humains. L'objectif ultime de la *RoboCup* n'est rien de moins que : « By 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer ». En plus de présenter un tournoi éliminatoire par ligue, cet événement compte également un symposium permettant à la communauté scientifique

rattachée d'échanger de précieuses connaissances. Le symposium est d'ailleurs une bonne occasion de présenter des travaux pertinents pour d'autres applications que le soccer robotisé. La communauté de la *RoboCup* permet de regrouper près de 4000 chercheurs provenant d'environ 40 pays.

Afin de permettre aux robots de se repérer sur le terrain de jeu, les différentes ligues définissent des balises et spécifications aux caractéristiques connues. Ainsi, dans la « Middle Size Robot League », deux équipes de 4 à 6 robots s'affrontent sur un terrain d'environ 8x12m où des balises bleu et jaune sont disposées aux 4 coins du terrain. Les buts et le ballon sont également de couleurs spécifiques. La figure qui suit présente une image d'un match typique de cette ligue.

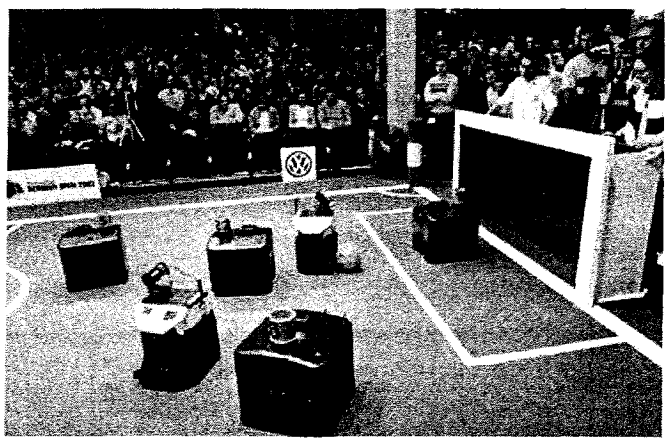


Figure 0.1.1: Image d'un match de la *RoboCup* « Middle Size Robot League ».

0.3 Définitions préalables

Afin de clarifier l'ensemble de la terminologie utilisée tout au cours du développement du présent mémoire, il est important de clarifier certains éléments par des définitions claires. Les définitions présentées dans cette section ont été tirées ou inspirées de définitions tirées du Grand dictionnaire terminologique (Office québécois de la langue française, [29]).

Dans un premier temps, le terme « agent » au sens large est souvent utilisé pour désigner un robot autonome. Voici donc une définition de ce terme permettant de vérifier le lien entre les deux termes.

Agent (déf.) :
Entité physique ou virtuelle possédant des ressources propres, capable de percevoir son environnement, d'agir sur lui, de communiquer directement avec d'autres agents et dont les comportements visent à satisfaire ses propres objectifs.

Selon cette définition du terme « agent », mis à part pour la communication directe, un robot peut être vu comme un agent autonome qui évolue dans un environnement physique. Ainsi, le domaine des systèmes multi-robots est intimement lié à celui des systèmes multi-agents au sein duquel il se fait de nombreuses recherches intéressantes. Le terme « système robotisé » au sens large est utilisé pour désigner tout type de robot, que ce soit un robot manipulateur, un robot mobile pris individuellement, ou encore un système multi-robots composé de plusieurs robots mobiles. Voici donc une définition acceptable de ce terme suivie de la définition de « système multi-robots coopératif », terme plus spécifique au présent projet.

Système robotisé (déf.) :
Machine automatique asservie, polyvalente et reprogrammable qui possède la flexibilité mécanique, la souplesse, l'adaptabilité et l'autonomie nécessaires, soit pour effectuer des tâches variées qui exigent des facultés propres à l'être humain à la fois sur les plans moteur et cérébral, soit pour remplir des fonctions motrices propres à ce dernier.

Système multi-robots coopératif (déf.) :
Système robotisé composé de plusieurs robots et possédant des mécanismes leur permettant de coopérer dans l'exécution d'un ensemble de tâches.

Le terme « cognition » permettra d'englober toute forme d'intelligence nécessaire à un système robotisé pour lui permettre d'accomplir ses tâches. En voici une définition du domaine de la psychologie humaine. Il est possible de remplacer le terme « être humain » par « système robotisé » afin d'obtenir une définition adéquate au présent sujet.

Cognition (déf.) :
Fonction complexe multiple regroupant l'ensemble des activités mentales (pensée, perception, action, volonté, mémorisation, rappel, apprentissage) impliquées dans la relation de l'être humain avec son environnement et qui lui permettent d'acquérir et de manipuler des connaissances (associations, rétroaction, traitement de l'information, résolution de problèmes, prise de décision etc.).

Note(s) :
La cognition dépend de structures ou de sous-systèmes complexes du système nerveux central. Bien que pour certains auteurs, le terme « cognition » désigne aussi bien les connaissances acquises que les processus permettant leur acquisition et leur manipulation, il est d'usage de réserver le terme à l'activité qui conduit à la connaissance, à l'exclusion des connaissances elles-mêmes.

Le terme « architecture » sera employé à maintes reprises dans le présent développement afin de désigner différentes structures et systèmes. Il est donc primordial d'en donner une définition claire et acceptable. La définition qui suit est très générale et même vague, mais elle permet la mise en forme de définitions plus spécifiques.

Architecture (déf.) :
Art de disposer les éléments d'un système, d'un édifice ou d'un appareil et de leur donner une forme harmonieuse.

Afin de désigner toute structure logicielle permettant à un système robotisé de fonctionner, le terme « architecture de contrôle » peut être employé. En voici une définition convenable.

Achitecture de contrôle (ou commande) (déf.) :
Structure organisée de l'ensemble des composantes logicielles nécessaires au bon fonctionnement d'un robot, composée de différentes couches ou niveaux. L'architecture de commande détermine la manière dont un robot réagit à son environnement, prend des décisions et organise ses actions en vue d'exécuter la tâche qui lui est attribuée.

Dans le cas d'un système multi-robots composé de robots autonomes mais travaillant en coopération, cette architecture peut être développée et étudiée de façon locale, pour un robot pris individuellement, et de façon globale, pour l'ensemble du système composé de plusieurs robots et de plusieurs autres composantes logicielles.

Note(s) :
Selon l'Office québécois de la langue française, le syntagme « architecture de contrôle » est à éviter pour désigner cette notion, car le terme « contrôle » utilisé dans ce sens est un anglicisme. Toujours selon l'OQLF, l'utilisation du terme *contrôle*, dans le sens de « commande », est fautive et constitue un faux ami. En effet, si le terme français *contrôle* désigne une opération de surveillance (*monitoring*), le terme anglais *control* implique une intervention et a comme équivalent *commande*. Néanmoins, considérant que le terme « commande » réfère plus spécifiquement à la commande d'un système asservi, et que le présent projet considère l'utilisation de plusieurs hiérarchies de contrôle. Le terme « contrôle » sera employé pour désigner l'ensemble des mécanismes, toute hiérarchie confondue, tandis que le terme « commande » fera référence au contrôle de premier ou deuxième niveau d'un système asservi.

Étant le thème principal de ce projet, le terme « architecture décisionnelle » se doit lui aussi d'être défini clairement. Il sera ainsi possible d'en comprendre le sens et de l'insérer justement dans le vaste domaine de la robotique. En voici donc une définition spécifique aux systèmes robotisés.

Achitecture décisionnelle (déf.) :
Structure organisée de l'ensemble des composants logiciels qui permet à un robot ou à un agent de prendre des décisions et qui définit les propriétés cognitives de ce dernier.

Pour qu'un système multi-robots coopératif puisse fonctionner convenablement, il lui faut généralement compter sur une certaine forme de communication. La structure des liens de communications peut varier d'une application à une autre et le terme « architecture de communication » sera employé pour désigner une telle structure. En voici une définition concise.

Architecture de communication (déf.) :
Structure organisée de l'ensemble des composants logiciels qui permet à un système robotisé d'échanger les informations nécessaires au bon fonctionnement de ce dernier.

Un système robotisé peut être composé de plusieurs niveaux ou hiérarchies de contrôle. Partant de l'asservissement de ses actuateurs aux mécanismes d'apprentissage et d'intelligence artificielle, les diverses hiérarchies présentes dans un système peuvent être nombreuses et variées. Dans le cas de tels systèmes, l'utilisation du terme « contrôle hiérarchisé » est appropriée et en voici une définition.

Contrôle hiérarchisé (déf.) :
Méthodologie de contrôle qui vise à modulariser sous forme de hiérarchie les différents niveaux de contrôle d'un système robotisé. La hiérarchie la plus basse touche directement aux actuateurs du système tandis que le nombre et le type des hiérarchies supérieures sont établis en fonction de la tâche à accomplir par ce dernier.

CHAPITRE 1 - SYSTÈMES MULTI-ROBOTS COOPÉRATIFS, ÉTAT DE L'ART

Les concepts reliés au développement des systèmes multi-robots coopératifs proviennent d'une grande variété de domaines. Il y a en premier lieu tous les domaines rattachés à la robotique mobile : mécatronique, théorie des systèmes, lois de commande, informatique embarquée, intelligence artificielle, systèmes de perception. En plus de ces domaines viennent s'ajouter ceux rattachés aux systèmes multi-robots : intelligence artificielle distribuée, systèmes multi-agents, systèmes de communication. Il serait peut-être inutile de faire un survol des éléments pertinents de chacun des domaines rattachés aux systèmes multi-robots coopératifs puisque la problématique du présent projet sera assez bien délimitée. Il est dans ce cas justifié de circonscrire les éléments abordés par le présent projet et d'en établir l'état de l'art.

Si l'objectif du présent projet est de développer une architecture décisionnelle bien adaptée à la problématique des systèmes multi-robots coopératifs, il est nécessaire d'étudier différentes architectures de contrôle pertinentes ainsi que d'étudier différents systèmes multi-robots expérimentaux. L'étude de Uny Cao et coll. ([42]) dresse un portrait assez détaillé de la robotique mobile coopérative tel qu'on la retrouvait vers le milieu des années 90. Depuis ce temps, des progrès fulgurants dans le domaine des technologies de l'information ont vu le jour. Les recherches sur les systèmes multi-robots coopératifs ont également progressé depuis ce temps. En effet, l'évolution impressionnante de la puissance de calcul des ordinateurs ainsi que la capacité accrue d'intégration de divers périphériques ont grandement facilité le développement de systèmes multi-robots expérimentaux démontrant diverses capacités coopératives. De plus, de nouvelles architectures de contrôle adaptées aux capacités actuelles des systèmes embarqués ont vu le jour.

1.1 Architectures de contrôle pour systèmes robotisés

Il existe de nombreux précurseurs aux diverses architectures développées aujourd'hui. La « Subsumption Architecture » de Brooks ([9]) ou encore la robotique dite « Behavior-Based Robotics » (Arkin, [6]) en sont de bon exemples. L'architecture *ALLIANCE/L-ALLIANCE* est également un très bon exemple de modèle couramment référencié encore aujourd'hui. Cette

architecture est présentée dans les lignes qui suivent de même que quelques architectures récentes, bien adaptées aux systèmes modernes, et qui rejoignent les objectifs du présent projet.

ALLIANCE/L-ALLIANCE : An Architecture for Fault Tolerant Multi-Robot Cooperation

L'architecture *ALLIANCE* (Parker, [31]) a été développée dans le but de permettre la coopération au sein de systèmes multi-robots hétérogènes. Elle se veut une architecture distribuée qui impose certaines règles d'implantation de façon à assurer une certaine tolérance aux fautes. La Figure 1.1 résume schématiquement le fonctionnement de cette architecture. L'architecture *ALLIANCE* implémente à un premier niveau les principes de la « Subsumption Architecture » (Brooks, [9]) par l'utilisation de comportements réactifs (fonctions reliant directement les actuateurs aux capteurs). Ensuite, à un deuxième niveau, le principe de *Behavior Set* est utilisé dans le but de contrôler l'activation ou l'hibernation de groupes de comportements situés au premier niveau. Ce deuxième niveau permet donc à l'architecture de se reconfigurer dynamiquement, ce qui permet d'adresser la problématique de la coopération de systèmes multi-robots hétérogènes. Pour arriver à établir une telle coopération, l'architecture utilise un troisième niveau, basé sur le principe de *Motivational Behavior*. Ce troisième niveau permet tout simplement d'activer/désactiver le deuxième niveau, celui des *Behavior Sets*, mais en plus de considérer les informations sensorielles comme le font les autres niveaux, il considère des informations de motivation interne permettant entre autre l'inhibition latérale des motivations (*cross-inhibition*), et finalement des informations provenant de la communication inter-robots.

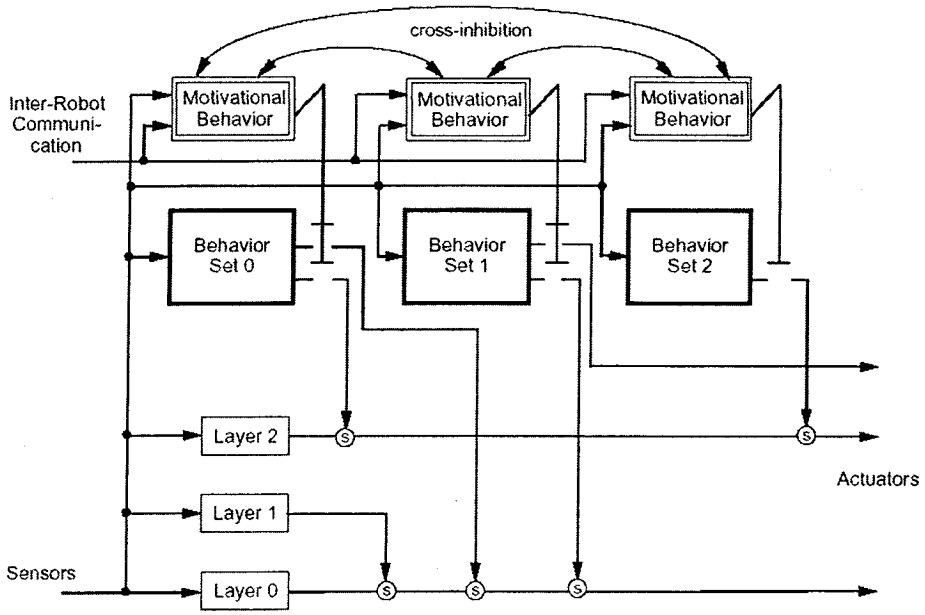


Figure 1.1 : Schéma résumant l'architecture *ALLIANCE*. (Tiré de Parker, [31])

Une extension de l'architecture *ALLIANCE*, nommée *L-ALLIANCE*, a été développée de façon à permettre l'implantation de mécanismes d'apprentissage par renforcement (« reinforcement learning ») au sein de l'architecture. En observant la Figure 1.2 on remarque la distinction avec l'architecture originale, qui correspond à l'ajout du principe de *Monitor*. Brièvement, le *Monitor* est en mesure d'influencer les comportements de la troisième couche, celle des *Motivational Behavior*.

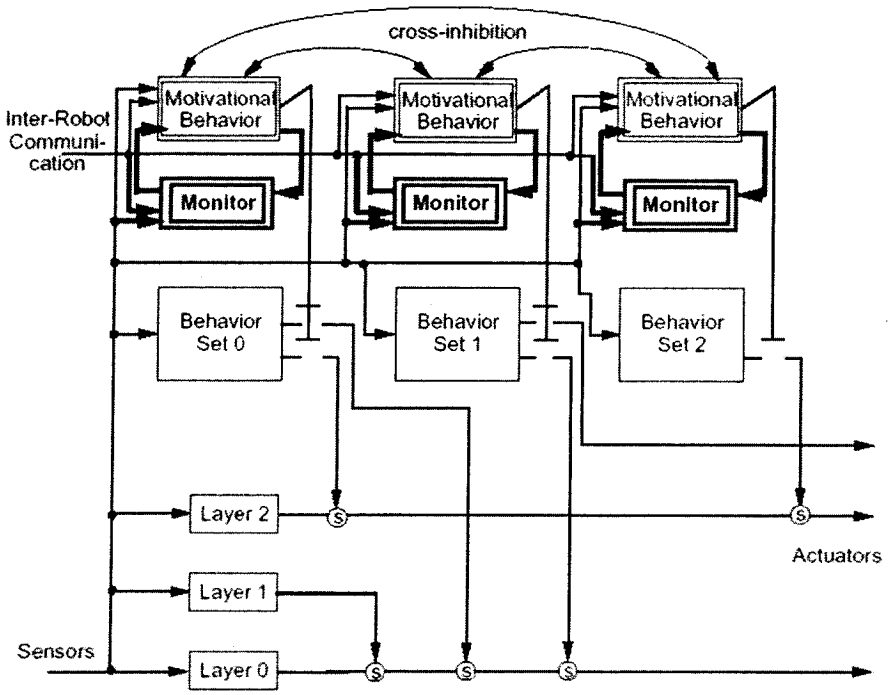


Figure 1.2 : Schéma résumant l'architecture *L-ALLIANCE*. (Tiré de Parker, [31])

CAMPOUT : Control Architecture for Multirobot Planetary OUTposts

CAMPOUT (Huntsberger et coll., [20]) est une architecture qui a été développée par la NASA au Jet Propulsion Laboratory du California Institute of Technology. L'architecture proposée par *CAMPOUT* se veut une architecture hybride (à la fois réactive et délibérative). *CAMPOUT* est à priori destinée pour les systèmes multi-robots à perception et cognition distribuées. Elle part du principe que chacun des robots est une entité indépendante et que la prise de décision se fait de façon entièrement, et strictement, distribuée.

Cette architecture est composée de trois couches distinctes (voir Gat, [18], pour plus de détails sur ce type d'architecture). La couche supérieure permet la planification, l'allocation et le monitorage hiérarchiques de tâches. La seconde couche permet la composition de différentes classes de comportements. La dernière couche de cette architecture permet de faire le lien entre la couche des comportements et le matériel du système robotisé (capteurs et actuateurs). La Figure 1.3 schématise le fonctionnement de l'architecture de *CAMPOUT*. On peut y retrouver les différentes classes de comportements ainsi que les différents outils disponibles pour chacune des tâches. Ces

outils semblent intéressants puisqu'ils offrent une interface évoluée pour définir la composition des éléments des différentes couches, soit par des langages de très haut niveau ou par outils graphiques.

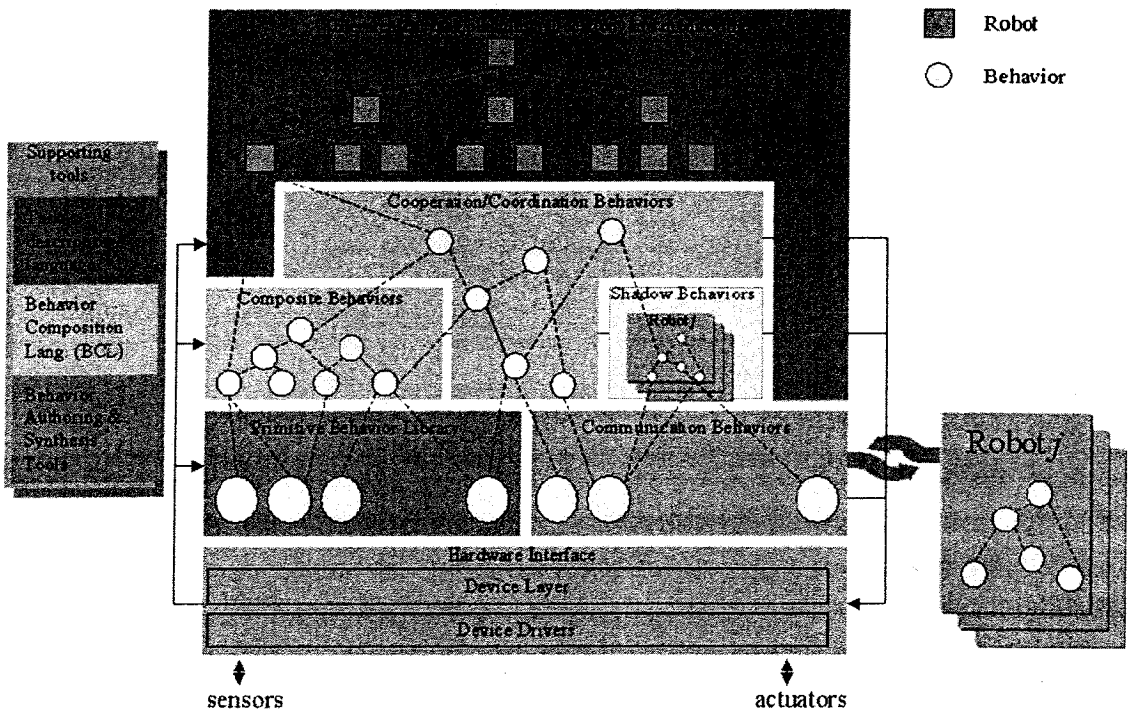


Figure 1.3 : Schéma résumant l'architecture *CAMPOUT*. (Tiré de Pirjanian et coll., [32])

L'architecture étant à la base conçue pour les systèmes multi-robots coopératifs, elle intègre certains éléments pertinents à ces systèmes, comme une infrastructure de communication (basée sur des sockets TCP/IP). Au niveau des comportements, trois classes sont importantes pour ces systèmes, soient les comportements de communication (*Communication Behaviors*), les comportements images (*Shadow Behaviors*) qui permettent la représentation des comportements des coéquipiers et finalement les comportements de coopération/coordination (*Cooperation/Coordination Behaviors*). Pris individuellement, les robots d'un système peuvent utiliser des comportements primitifs et composés (*Primitive Behaviors* et *Composite Behaviors*) pour accomplir leur travail. La gestion de l'utilisation des divers comportements est assurée par la couche supérieure de l'architecture (*Hierarchical task planning, allocation and monitoring*).

L'architecture *CAMPOUT* a été utilisée sur divers systèmes robotisés développés au JPL dans le cadre de développements en robotique mobile liés à l'exploration extra-planétaire. Les robots *SRR*, *FIDO* et *LEMUR* ont entre autre été interfacés par cette architecture. Un scénario de montage de tente photovoltaïque en environnement martien est utilisé comme banc d'essai afin de valider l'utilisation de *CAMPOUT* pour ce type de mission. Dans ce scénario, deux robots ont entre autre la tâche de transporter en coopération des conteneurs de matériel.

CLARAty : Coupled Layer Architecture for Robotic Autonomy

CLARAty (Nesnas et coll., [27]) a également été développée par la NASA au Jet Propulsion Laboratory du California Institute of Technology. L'objectif principal derrière *CLARAty* est de développer une architecture pour des composantes robotiques génériques et réutilisables et qui peut être adaptée à diverses plateformes robotisées hétérogènes. À priori, cette architecture ne traite aucunement des systèmes multi-robots, mais les objectifs de développement ainsi que les concepts développés en font une architecture intéressante à étudier et elle peut toujours être étendue pour les systèmes multi-robots. Son implantation actuelle utilise le langage C++ avec une approche orientée-objet.

Le concept de cette architecture innove par rapport aux plus traditionnelles architectures à trois couches. Un schéma résumant cette architecture est présenté à la Figure 1.4. On y remarque seulement deux couches : la couche fonctionnelle (*Functional Layer*) et la couche décisionnelle (*Decisional Layer*). La couche fonctionnelle définit une multitude de composantes génériques réutilisables pour interfacer et contrôler divers robots et diverses composantes de systèmes robotisés. Grâce à l'utilisation de patrons de conception pertinents en programmation par objet, les composantes définies sont hautement modulaires. La couche fonctionnelle cumule les rôles de planification et d'exécution (deux couches distinctes généralement). La couche décisionnelle considère les ressources du système ainsi que les objectifs de mission pour planifier, céduler et exécuter des plans d'activité. L'interaction entre les deux couches se fait par une approche client-serveur et elle est bidirectionnelle et possible à tous les niveaux hiérarchiques.

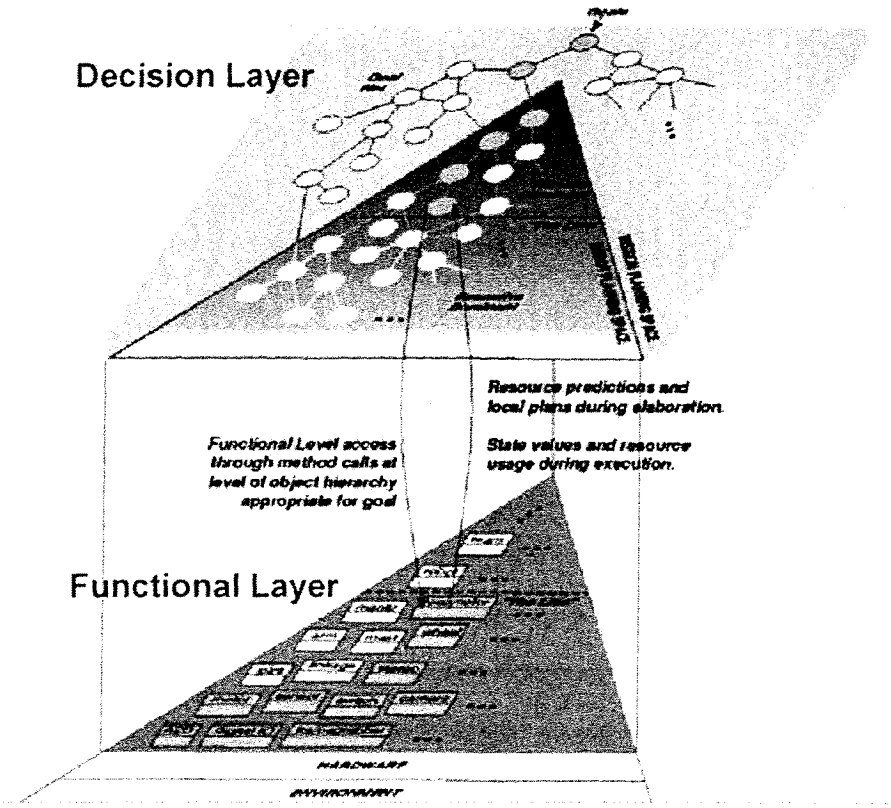


Figure 1.4 : Schéma résumant l'architecture CLARAty. (Tiré de Nesnas et coll., [27])

L'architecture *CLARAty* a été testée sur de nombreuses plateformes de robots de la NASA. Son bon fonctionnement et ses caractéristiques modulaires et génériques ont été démontrés sur les robots suivants : *ATRV*, *Rocky 7*, *Rocky 8*, *FIDO*, *K9*, *Sojourner*, et *Hyperion*.

XABSL : Extensible Agent Behavior Specification Language

Voici un excellent exemple de retombées de travaux effectués dans le cadre la RoboCup. L'architecture *XABSL* (Lötzsch et coll., [25]) a été développée et utilisée depuis quelques années par l'équipe *GermanTeam* de la « Sony 4-Legged Robot League ». En utilisant cette architecture, cette équipe a remporté les éditions 2004 et 2005 de la Coupe du Monde en plus de très bien se classer dans les challenges techniques depuis plusieurs années.

Brièvement, *XABSL* est un dialecte basé sur le langage XML permettant de développer des comportements pour agents autonomes. Un agent y est considéré comme étant un regroupement hiérarchique de modules de comportements appelés *options*. La prise de décision au sein des

options se fait grâce à des machines à états. La hiérarchie peut se représenter graphiquement et est toujours terminée par des *basic behaviors*. Un exemple d'*option* développée pour l'équipe *GermanTeam* est présenté à la Figure 1.5 puis une transition d'état propre à cette option est présentée à la Figure 1.6.

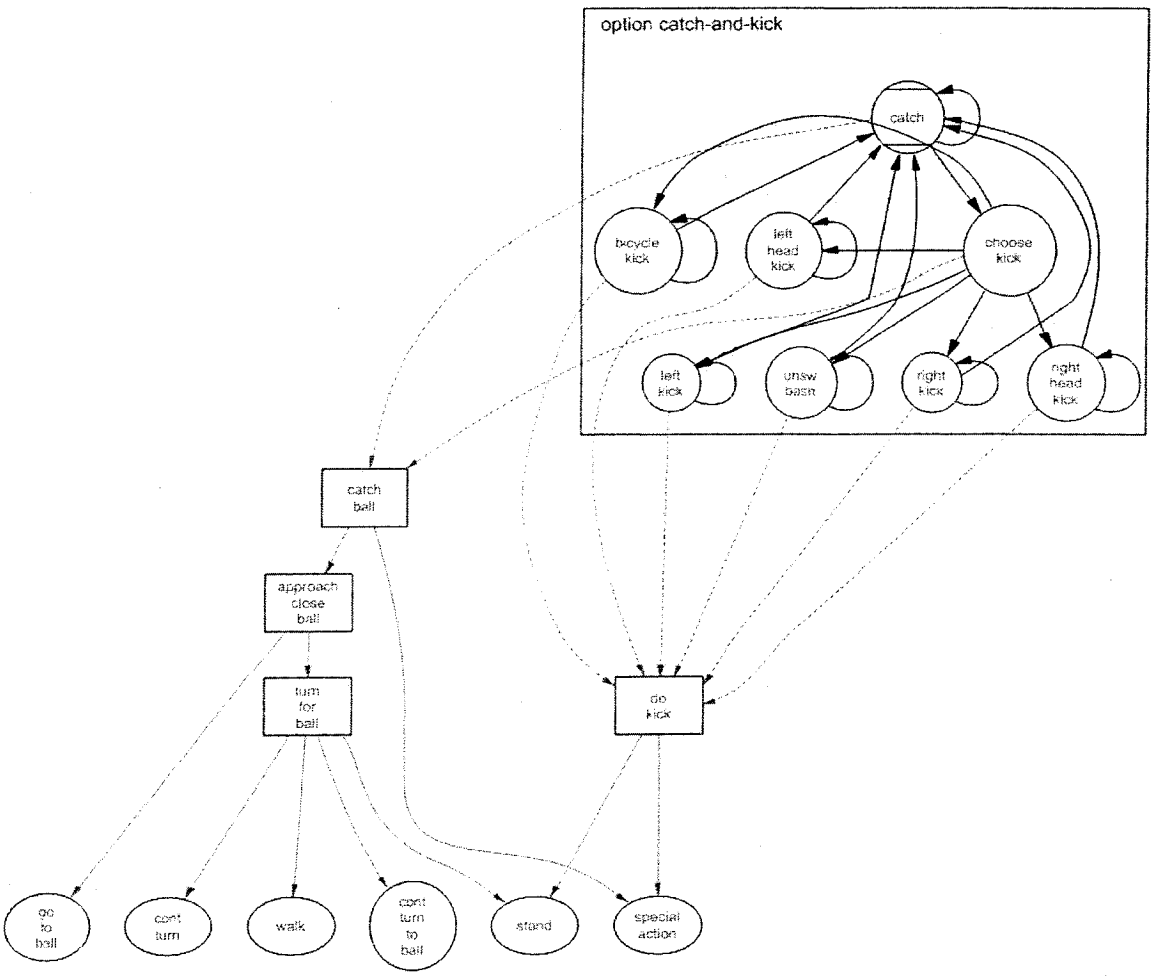


Figure 1.5 : Exemple d'option, *catch-and-kick*, définie au sein de l'architecture *XABSL*. (Tiré de Lötzsch, [24])

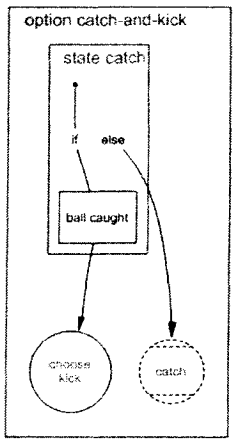


Figure 1.6 : Exemple de transition d'état définie au sein de l'architecture *XABSL*. (Tiré de Lötzsch, [24])

Cette architecture est conceptuellement très simple, mais efficace. L'usage d'un langage de haut niveau comme le XML comporte de nombreux avantages puisque ce langage permet de se détacher complètement de l'implantation matérielle du système robotisé. Les comportements de plus bas niveaux peuvent alors être programmés dans divers langages, ce qui rend l'architecture hautement modulaire et générique. De plus, les outils graphiques associés et la simplicité du dialecte facilitent grandement le développement et la documentation de systèmes décisionnels. Le fait d'utiliser des machines à états pour la prise de décision peut cependant être contraignant pour l'implantation de mécanismes de décision arbitraires.

1.2 Systèmes multi-robots coopératifs

Il existe de nombreux projets de recherche portant sur la mise en fonction de systèmes multi-robots coopératifs. Quelques exemples concrets de tels projets sont présentés dans les paragraphes qui suivent. Une attention particulière est portée à l'exploration extra-planétaire.

Robots pour exploration extra-planétaire

L'exploration extra-planétaire constitue pour la robotique une avenue de développement fascinante. Il n'est pas surprenant de voir les ressources considérables déployées par les agences spatiales internationales pour le développement de diverses technologies robotiques. Par exemple, du budget annuel total de la NASA d'environ 16 milliards de dollars U.S., près du tiers est employé aux missions d'exploration robotisées et au développement de technologies pour de

futures explorations humaines/robotiques (Aldridge et coll., [5]). Dans les années à venir ces chiffres sont appelés à croître considérablement. Le plan d'exploration du gouvernement américain vise dans un premier temps le développement de missions habitées pour la Lune vers 2015 dans le but de développer des technologies pour des missions habitées pour Mars vers 2030. La robotique jouera un rôle majeur dans le développement de ces missions et selon l'Agence Spatiale Canadienne plusieurs technologies développées ici seront utilisées (ASC, [2]).

Avec un temps de transmission de près de 40 minutes aller-retour Terre-Mars, la téléopération est très peu souhaitable pour un système robotisé déployé sur Mars. L'autonomie des systèmes développés constitue donc un élément-clé pour le succès des missions robotiques martiennes. Ainsi, plusieurs travaux de recherche se font sur les architectures de contrôle à capacités délibératives, *CAMPOUT* et *CLARAty* en sont de bons exemples, mais il y a également plusieurs systèmes expérimentaux simulant des systèmes robotisés entièrement autonomes en mission sur Mars.

Un exemple de projet de recherche expérimentale en ce sens est le « Robot Work Crew » développé par la NASA (Trebi-Ollennu, [41]). Ce projet adresse la problématique de la coopération multi-robots pour l'exécution de tâches en couplage rigide (manipulation coopérative) ainsi qu'à la coopération de groupes homogènes et hétérogènes de robots, toujours en considérant un environnement difficile comme le sol martien. Leur principale réalisation expérimentale est la collaboration de deux robots mobiles pour le déploiement d'une tente photovoltaïque en utilisant l'architecture *CAMPOUT* (Huntsberger et coll., [20]).

La construction de sites habitables par des systèmes robotisés est donc un sujet d'étude pertinent, mais l'exploration et l'extraction de ressources le sont également. En effet, une fois un site habitable établi, les missions de reconnaissance de l'environnement ainsi que l'approvisionnement en ressources utilisables sont essentiels au maintien d'une mission habitable. Plusieurs projets traitent de ces problèmes, par exemple le projet d'exploration robotisée du désert chilien d'Atacama (Carnegie Mellon University, [10]) qui constitue un environnement extrêmement aride semblable à celui de la surface martienne. Ce projet ne traite pas de systèmes multi-robots, mais plutôt de technologies-clés pour l'exploration et la navigation autonome dans le but de rechercher de la vie en environnements extrêmes, l'astrobiologie. La Figure 1.7 présente

une image du robot *Zoë* développé dans el cadre de ce projet. Ce robot est entre autre muni d'un système d'imagerie à fluorescence, d'un spectromètre et d'une laboureuse.

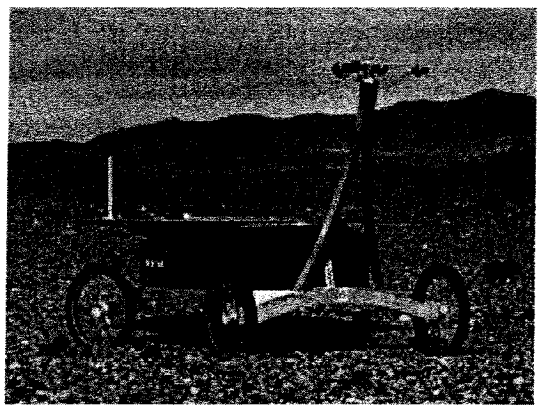


Figure 1.7 : Robot *Zoë* développé pour le projet d'exploration du désert d'Atacama. (Tiré de CMU, [10])

Soccer robotisé

Comme il a déjà été mentionné, le soccer robotisé constitue une plate-forme de développement de prédilection pour les systèmes multi-robots coopératifs. Les technologies qui en découlent sont souvent d'intérêt pour d'autres domaines d'applications. Afin de présenter un système typique développé pour le jeu de soccer robotisé, l'équipe *FU-Fighters* ([35]) de l'université FU Berlin sera prise en exemple. Cette équipe est présente dans la « Small Size Robot League » où elle est championne du monde en 2004 et 2005, ainsi que dans la « Middle Size Robot League » où elle se rapproche d'un sacre avec une $4^{ème}$ et une $2^{ème}$ places en 2004 et 2005 respectivement.

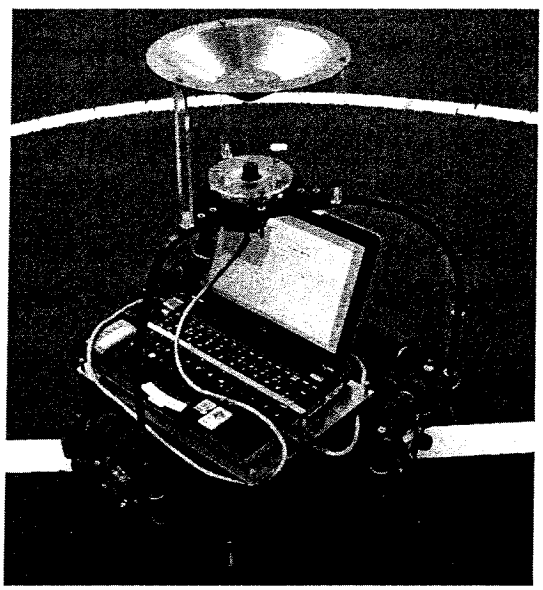


Figure 1.8 Robot de l'équipe *FU-Fighters* pour la « Middle Size Robot League ». (Tiré de Fu-Fighters, [35])

L'apport scientifique principal apporté par cette équipe se situe au niveau mécatronique ainsi qu'au niveau de la vision artificielle. Mais étant données leurs performances en compétition, nul doute que leur architecture de contrôle est adéquate! Ils semblent utiliser plusieurs concepts similaires tant pour leurs robots de la SSL que pour ceux de la MSL, ce qui indiquerait un aspect générique et modulaire intéressant de leur architecture. Leur concept d'architecture de contrôle (Egorova, [15]) est nommé « Extended Dual Dynamics ». La Figure 1.9 donne un schéma général du fonctionnement de l'architecture. Elle est principalement constituée de couches munies de 3 éléments : des capteurs (*Perceptual Dynamics*), des comportements (*Activation Dynamics*) et des actuateurs (*Target Dynamics*).

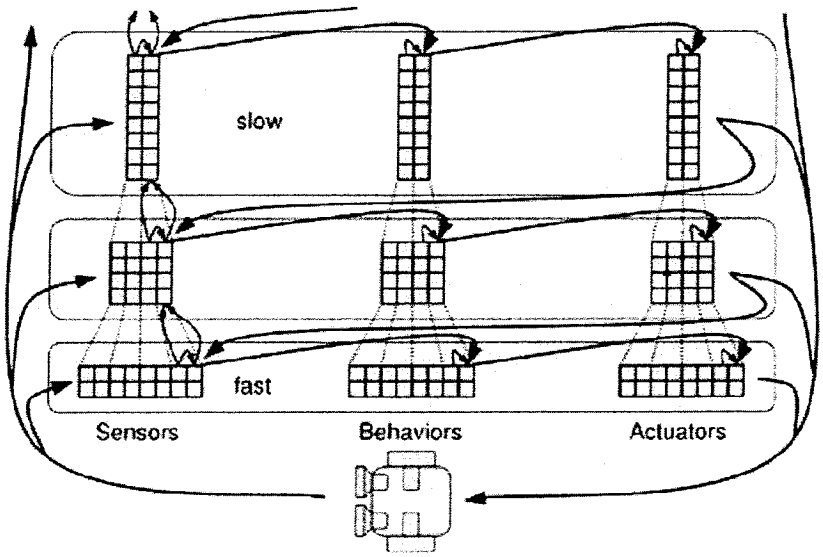


Figure 1.9 : Architecture « Extended Dual Dynamics » de l'équipe *FU-Fighters*. (Tiré de Egorova, [15])

L'architecture est dite strictement réactive et n'utilisant aucun modèle explicite du monde. Ainsi elle comporte seulement une hiérarchie de comportements, utilisant strictement les informations sensorielles comme entrées. L'inhibition possible des comportements de plus bas niveau par les comportements de plus haut niveau permet un contrôle hiérarchisé. Finalement, pour gérer le systèmes multi-robots coopératif, un étage au-dessus de tous les autres est utilisé, étage considérant les informations sensorielles de tous les robots et capable d'inhibition sur le groupe également. La Figure 1.10 présente cette structure.

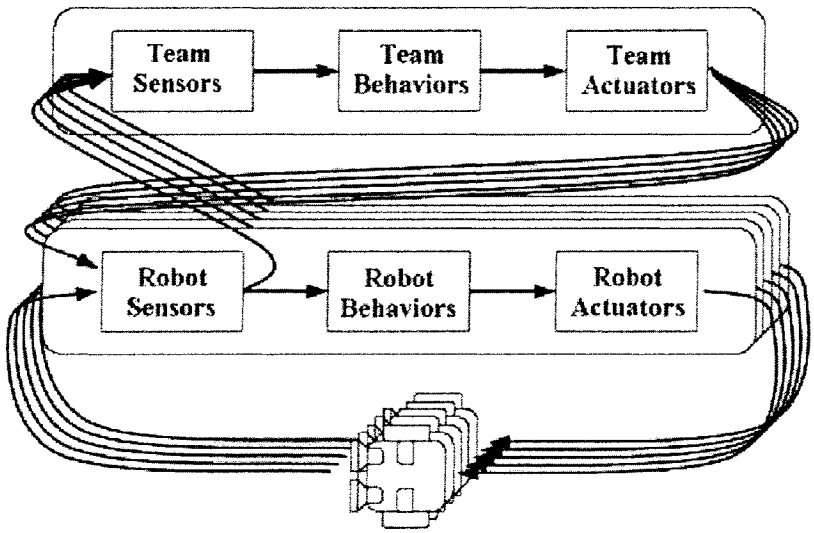


Figure 1.10 : Couche de comportements d'équipe pour l'architecture de l'équipe *FU-Fighters*. (Tiré de Egorova, [15])

Systèmes multi-robots au service de l'homme

Rares, voire inexistants, sont les systèmes multi-robots déployés en milieu habité par l'être humain dans le but de l'aider dans l'exécution de tâches, par exemple en milieu industriel. Si l'on se fie cependant aux recherches pour de telles applications, des systèmes multi-robots pourraient apparaitre dans notre vie quotidienne dans un avenir non si lointain. En effet, plusieurs projets de recherche traitent d'applications bien concrètes et présentent des résultats expérimentaux prometteurs. Par exemple, les systèmes robotisés pour l'agriculture présentent un potentiel intéressant (Pilarski et coll., [32], Noguchi et coll., [28]). Ou encore les autoroutes autonomes qui sont en développement depuis déjà plusieurs années, les travaux de l'AHSRA ([1]) au Japon étant en quelque sorte la référence dans ce domaine. Et de nombreuses autres applications en milieu humain sont imaginées et en développement.

Pour illustrer un bon exemple de système multi-robots coopératif au service de l'homme, le projet *Martha* (Alami et coll., [3]) sera utilisé. L'objectif de ce projet est de développer des techniques d'opération et de contrôle d'une flotte de robots mobiles autonomes pour le transport de conteneurs dans les ports maritimes, les aéroports et les cours de triage. En plus de faire diverses simulations dans des environnements de travail réels avec de larges groupes de robots, des expérimentations ont été menées sur une famille de 3 robots mobiles nommée *Hilare*.

L'architecture de contrôle derrière le projet *Martha* est basée sur des travaux de la même équipe (Alami et coll., [4]). La Figure 1.11 schématise le fonctionnement de cette architecture.

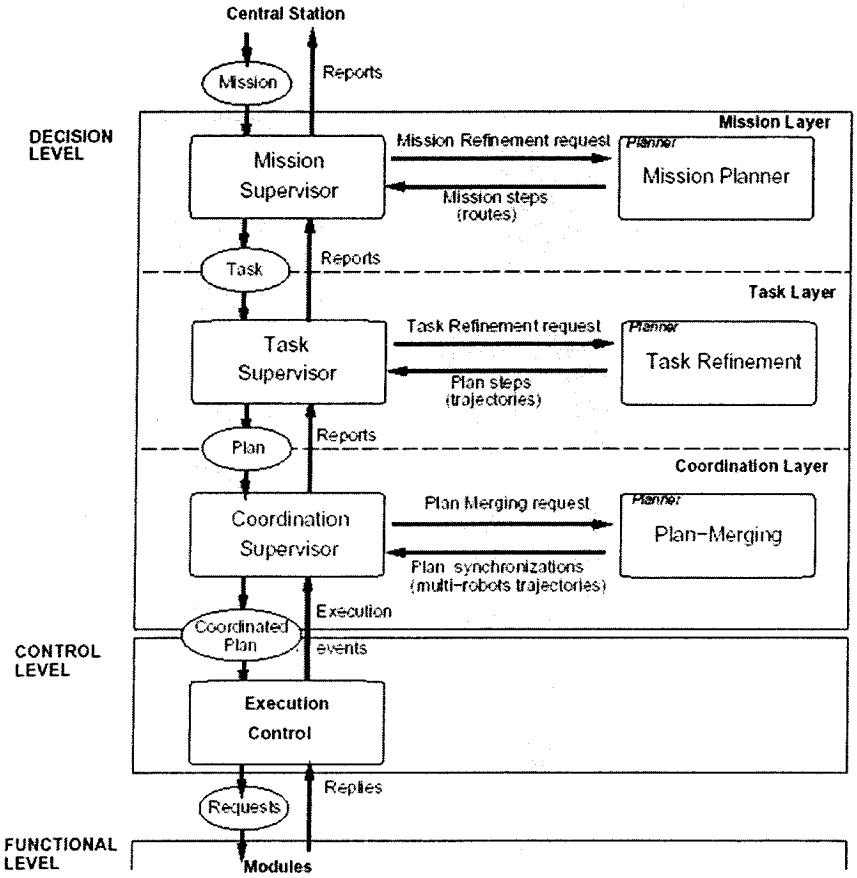


Figure 1.11 : Architecture de contrôle derrière le projet Martha. (Tiré de Alami et coll., [4])

On y retrouve trois niveaux, ce qui constitue donc une architecture à trois couches conventionnelle : le niveau fonctionnel au plus bas, le niveau de contrôle au milieu et le niveau décisionnel au haut de la hiérarchie. Le niveau décisionnel est séparé en trois étages : un premier au bas pour la gestion de la coordination avec les autres robots (*Coordination Layer*), un second pour la gestion de l'exécution de tâches (*Task Layer*), puis un dernier en haut pour la gestion de l'exécution de missions (*Mission Layer*). Chaque étage est composé d'une relation superviseur-planificateur afin de gérer et de raffiner l'exécution. Au-dessus de cette architecture se trouve la *Central Station* qui se charge seulement et strictement d'établir la mission à accomplir par chacun des robots, beaucoup d'autonomie décisionnelle étant laissée donc au système distribué. La mission reçue du *Central Station* est ensuite décomposée sous forme de tâches par le *Mission*

Layer. Chaque tâche est ensuite décomposée en plans par le *Task Layer*, puis ces plans sont modifiés en plans coordonnés par le *Coordination Layer*. Pour assurer la bonne coordination des plans, donc la coordination de l'équipe de travail, l'étage de coordination utilise le concept de *Plan-Merging Protocol*. Par ce protocole qui utilise une communication inter-robot, un robot est en mesure de fusionner son plan avec ceux des autres robots et donc de proposer un nouveau plan coordonné. Les plans sont finalement exécutés grâce au niveau de contrôle et au niveau fonctionnel.

Expérimentations en laboratoire

Il existe également plusieurs projets de recherche présentant des expérimentations en laboratoire sur des problèmes plus spécifiques reliés aux systèmes multi-robots coopératifs. L'objectif de ces projets va donc être de développer une solution au problème spécifique posé, le tout étant généralement inséré dans un cadre plus général de développement visant à plus long terme un transfert vers des applications concrètes.

Voici quelques thèmes intéressants et fréquemment étudiés expérimentalement. Certains de ces thèmes sont d'ailleurs souvent traités conjointement au sein d'un même projet. Un thème qui revient fréquemment est la navigation autonome d'une flotte de robots en formation, par exemple en utilisant un « leader » (Carpin et Parker, [11]) et en suivant une formation qui peut être déterminée dynamiquement (Lemay et coll., [23]). La coordination des actions de chacun des robots est alors cruciale. L'assignation dynamique de rôles (Chainowicz et coll., [12], Emery et coll., [16]) est également un thème très intéressant puisque dans ce cas la prise de décision, qui peut être distribuée, se doit d'être très bien synchronisée. Le problème de manipulation d'objets en coopération (Chainowicz et coll., [13], Song et Kumar, [38]) est également intéressant étant donnée le couplage rigide entre les robots et donc un besoin de coordination.

La majorité des systèmes expérimentaux mentionnés vont de façon générale présenter des implantations de mécanismes de contrôle relativement simples afin de répondre aux besoins efficacement, sans donc nécessiter toutes les fonctionnalités et caractéristiques d'une architecture développée dans un cadre général. Les développements de solutions pratiques au niveau des systèmes de perception et de la mécatronique ont tendance également à freiner le développement de mécanismes cognitifs de haut niveau. Néanmoins, tant au niveau conceptuel qu'au niveau

pratique des progrès sont constamment établis, ce qui permet donc graduellement d'améliorer les capacités des systèmes expérimentaux par le transfert et la mise en pratique d'éléments conceptuels.

CHAPITRE 2 - PROBLÉMATIQUE : ARCHITECTURE GÉNÉRIQUE POUR SYSTÈMES MULTI-ROBOTS

Le chapitre précédent donne donc une revue de différentes architectures et quelques systèmes multi-robots expérimentaux. On peut observer plusieurs similitudes parmi les éléments présentés, comme par exemple l'utilisation de mécanismes de contrôle hiérarchisé et l'utilisation de la communication comme outil de coopération. Malgré ces similitudes, il en ressort des différences majeures au niveau de la structure comme tel des architectures de contrôle. Chacune de ces architectures semble dévier à différents niveaux d'un modèle commun permettant de généraliser certains concepts. De plus, au niveau de l'implantation sur des systèmes expérimentaux, les différences sont également importantes. Chacun des systèmes expérimentaux répondant à un ensemble de tâches différent et utilisant des éléments matériels et logiciels bien différents, il en résulte des implantations qui sont spécifiques et peu génériques. Il est donc difficile, à partir de cette revue, d'établir un certain formalisme permettant de définir les éléments constitutifs d'un système multi-robots coopératif, les interrelations structurales entre ces éléments, ainsi que des méthodes génériques d'implantation sur des systèmes réels.

À priori, il n'était pas nécessairement prévu de développer une architecture comme tel car la possibilité d'utiliser une architecture déjà développée, suffisamment documentée et dont les sources étaient disponibles était envisagée. Cette possibilité a été abandonnée en cours de route pour différentes raisons. Premièrement, il faut remarquer qu'il existe une multitude d'architectures différentes qui répondent à des besoins variés. Rares sont celles, voire inexistantes, qui répondent entièrement aux besoins du présent projet. De plus, en termes de temps de développement, il est difficile d'évaluer la quantité de travail à effectuer pour se familiariser avec une architecture existante et développer les modules informatiques permettant de la tester. Dans le cas où les essais avec une architecture donnée s'avèrent insatisfaisants, il faut recommencer avec une autre et donc il peut être impossible d'en trouver une parfaitement bien adaptée, et si jamais elle existe le temps de développement pour arriver à l'intégrer peut être beaucoup plus grand que le temps mis pour développer une architecture répondant spécifiquement aux besoins établis. Néanmoins, l'étude des différents projets présentés préalablement permet d'élargir les connaissances sur les systèmes multi-robots coopératif et leurs

architectures de contrôle et de définir plus clairement les caractéristiques recherchées de l'architecture à développer.

Les paragraphes qui suivent vont présenter le point de départ pour le développement d'une architecture décisionnelle, c'est-à-dire tout ce qui peut être établi à priori permettant de motiver le développement d'un certain type architecture.

2.1 Objectifs et spécifications

Le but du présent projet est en quelque sorte d'établir certaines bases pour le développement d'un système multi-robots autonome et coopératif. Un tel système doit évidemment être composé d'une multitude de sous-systèmes mécatroniques, de nombreux modules logiciels variés et bien plus. Établir les bases pour le développement d'un tel système signifie en quelque sorte :

- Identifier l'ensemble des modules et sous-systèmes nécessaires au fonctionnement du système
- Modulariser le plus possible l'ensemble de ces éléments, en identifier clairement les divisions et les interrelations
- Établir un ensemble de concepts génériques et développer des modèles génériques pour chacun des modules du système
- Valider les développements effectués par diverses implantations sur des systèmes expérimentaux

Le point le plus important dans cette liste est peut-être le développement de concepts et de modèles génériques. En effet, la « mise en banque » de tels concepts et modèles permet de créer une base de connaissances et une librairie d'outils étant réutilisables lors de développements futurs. La pérennité de l'environnement de développement en est ainsi grandement facilitée.

L'élément le plus générique d'un système multi-robot est certainement son architecture de contrôle puisqu'elle définit de façon strictement conceptuelle l'ensemble des modules nécessaires au contrôle du système et leurs interrelations. Au sein de cette architecture se trouvent trois modules jouant des rôles fondamentaux : la perception, la cognition et le contrôle. Parmi ces trois modules, un seul peut être totalement indépendant du matériel, de la plate-forme mécatronique développée. En effet, malgré plusieurs concepts génériques possibles au niveau de la perception

et du contrôle, le développement de ces modules est influencé par les aspects physiques du système. Par contre, le module de cognition se trouve à jouer un rôle central et indépendant des aspects physiques. L'utilisation de modèles génériques au sein de ce module est donc particulièrement intéressante puisque ces modèles peuvent être réutilisés pour répondre aux besoins d'une multitude de systèmes multi-robots accomplissant des tâches différentes.

Dans le cadre des développements effectués à l'École Polytechnique préalablement à ce projet, la majorité des éléments nécessaires à la mise en fonction d'un premier système multi-robots autonome ont été développés (Robofoot ÉPM, [36]), mais aucune architecture décisionnelle n'avait encore été développée spécifiquement pour ce système.

Ainsi, l'objectif principal de ce projet est de :

1. Développer une architecture décisionnelle spécifiquement conçue pour des systèmes multi-robots contrôlés en temps-réel, donc munie de mécanismes pour aider à la coopération.

Il est également désiré de :

2. Valider le bon fonctionnement de l'architecture développée en la testant sur un premier système multi-robots autonome et coopératif.

Le système multi-robots en question sera une équipe de robots joueurs de soccer pour la « Middle Size Robot League » de la *RoboCup* (RoboCup Federation, [34]).

Avant de se lancer dans des détails plus conceptuels, il est nécessaire de spécifier plus précisément quelles seront les fonctionnalités et caractéristiques visées par l'architecture développée. Ainsi, voici une liste des spécifications fonctionnelles que l'architecture cherchera à respecter :

- Obtenir une architecture décisionnelle pour systèmes multi-robots coopératif bien adaptée pour un large éventail d'applications.
- Possibilité d'utilisation sur des systèmes multi-robots homogènes ainsi que des systèmes hétérogènes.
- Possibilité d'utilisation sur des systèmes à processus cognitifs centralisés et/ou distribués.
- Architecture décisionnelle permettant de fonctionner de façon distribuée sans communication explicite.

- Si la communication explicite est possible par lien de communication, l'utiliser efficacement en augmentant les capacités de coopération du système, tout en limitant la quantité de données en transmission.
- Développer une structure décisionnelle hiérarchique au nombre de hiérarchies arbitraire.
- Pour la prise de décision hiérarchisée, permettre tout mécanisme décisionnel arbitraire.
- Permettre la supervision décisionnelle, c'est-à-dire la possibilité de surcharger les décisions prises de façon distribuée par un mécanisme central.
- Obtenir une architecture décisionnelle dont la structure modulaire et la représentation graphique tendent à faciliter la supervision décisionnelle ainsi que la définition et la révision de la structure décisionnelle.
- Implanter l'architecture à un niveau logiciel générique de façon à permettre le développement multi-plateforme.

Ces spécifications serviront donc de lignes directrices pour le développement de l'architecture.

2.2 Caractère de l'architecture recherchée

Lorsque vient le temps de développer les capacités cognitives d'un système robotisé, il est difficile, voire indispensable, de ne pas considérer l'intelligence humaine comme un modèle intéressant à étudier. Le but du présent projet n'est pas de traiter d'intelligence artificielle et encore moins de mimer l'intelligence humaine. Cependant, notre intelligence demeure une merveille et un mécanisme encore aujourd'hui mal compris tant il est complexe. L'être humain peut certainement être considéré comme le meilleur exemple de système autonome intelligent identifié à ce jour par la science. Mais puisque nous ne connaissons pas encore tous les détails précis des mécanismes cognitifs de l'intelligence humaine, il est encore impossible de la mimer. Néanmoins, il est possible d'établir certaines observations sur son fonctionnement et de s'en inspirer dans le but d'obtenir un système autonome possédant des capacités d'adaptation et d'apprentissage satisfaisantes. Le présent travail n'aura donc pas comme objectif de développer une architecture permettant d'implanter un modèle d'intelligence humaine, mais il pourra s'inspirer de certains faits de façon à obtenir une architecture suffisamment versatile pour aspirer à l'implantation de modèles cognitifs pertinents pour systèmes robotisés autonomes.

Afin de bien situer l'architecture développée au sein des développements scientifiques reliés au domaine des architectures pour systèmes robotisés autonomes, il est nécessaire de bien définir le type d'architecture recherché. R.C. Arkin ([6]) présente une analyse assez détaillée des différents types d'architectures possibles en en résumant les principales caractéristiques. La Tableau 2.1, tiré de son livre, présente de façon synthétisée le spectre des architectures de contrôle pour robots mobiles. On y remarque deux extrêmes. L'un étant l'architecture strictement délibérative, capable de mécanismes cognitifs évolués mais étant fortement dépendante de représentations de l'environnement, présentant une réponse lente et une latence variable. L'autre extrême, l'architecture strictement réactive, qui permet une réponse temps réel, déterministe dûe à son traitement simple, indépendante de l'environnement (absence de modèles), mais qui ne peut qu'implanter des mécanismes cognitifs très peu évolués.

Délibératif	*Réactif*
Purement symbolique	**Réflexif**
Vitesse d'exécution	
Capacités prédictives	
Dépendance à des modèles précis et complets de l'environnement	
Dépendance à la représentation	**Indépendance à la représentation**
Réponse lente	**Réponse temps-réel**
Intelligence haut niveau (cognitive)	**Intelligence bas niveau**
Latence variable	**Traitement simple (déterminisme)**

Tableau 2.1 : Spectre des architectures de contrôle pour robots mobiles (Traduit de Arkin, [6]).

Par le passé, des travaux ont porté sur des architectures aux deux extrêmes. Par exemple l'architecture « Subsumption Architecture » de Rodney Brooks ([9]) qui présentait un contraste marqué par rapport aux travaux de l'époque traitant principalement d'intelligence artificielle et de mécanismes strictement délibératifs. Par la suite, depuis environ une dizaine d'années, une nouvelle tendance fit son apparition, tentant de démontrer par des analogies biologiques ou encore par expérimentation, que les architectures hybrides, capables de mécanismes délibératifs, mais démontrant également des capacités réactives, peuvent offrir en quelque sorte le meilleur

des deux mondes. Aujourd'hui, la majorité des architectures développées sont hybrides. La définition suivante peut être formulée pour une architecture hybride.

Achitecture hybride (déf.) :
Architecture de contrôle pour système robotisé jumelant capacités délibératives et capacités réactives, souvent sous forme hiérarchisée, permettant au système d'étendre sa capacité d'interaction avec l'environnement.

Par analogie avec la biologie humaine, les capacités réactives du robot correspondent aux réflexes de l'être vivant, c'est à dire des mécanismes réactifs à réponse très rapide. L'être humain étant capable d'interpréter une énorme quantité d'information et de formuler des plans sur des horizons de temps très variables, ses capacités cognitives doivent inspirer le développement de telles capacités pour systèmes autonomes. Ainsi, un robot doit être en mesure de réagir rapidement lorsqu'il est placé dans un environnement hautement dynamique. Il se doit donc d'être muni de capacités réactives adéquates. De plus, les tâches à effectuer par un système robotisé peuvent s'échelonner sur un horizon de temps relativement long et elles peuvent dépendre d'interrelations concrètes avec l'environnement extérieur. La capacité d'un robot à modéliser son environnement ainsi qu'à établir des plans, de complexité et de durée variables, peut donc l'aider à effectuer des tâches plus complexes.

L'architecture décisionnelle développée se voudra donc hybride, munie de capacités délibératives et réactives. Pour ce qui est des capacités réactives, les comportements du système robotisé, elles seront ancrées au dessus d'un étage de contrôle inférieur qui prend en charge l'asservissement des actuateurs et qui permet d'agir sur l'environnement extérieur. Puis, les capacités délibératives, les mécanismes décisionnels, se situeront à une couche supérieure, au-dessus de la couche réactive des comportements. La Figure 2.1 présente ce modèle d'architecture, qui peut être vu comme un traditionnel modèle à trois couches (Gat, [18]). Une première couche permet au système de gérer la commande de ses actionneurs par divers contrôleurs. Un seconde couche, la couche réactive, permet de générer des actions par le biais de divers comportements. Finalement, la troisième et dernière couche, la couche délibérative, permet une prise de décision hiérarchisée, avec un nombre d'étages et ainsi un degré de délibération variables.

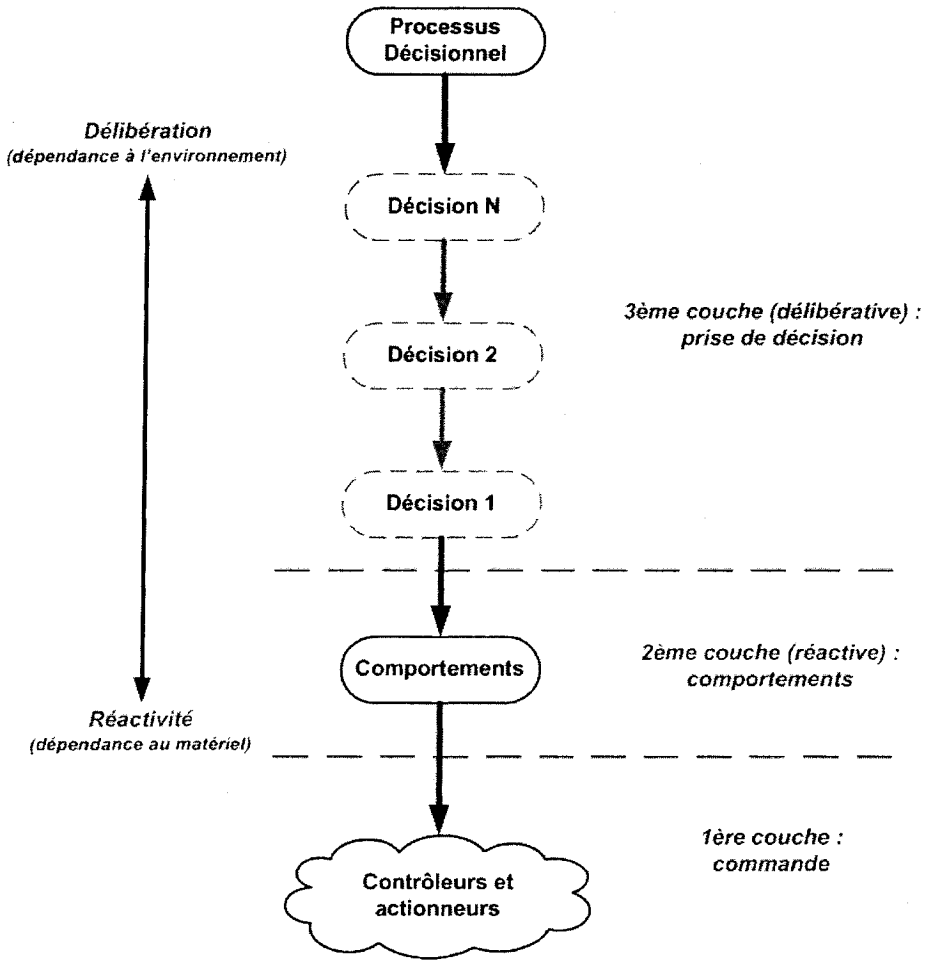


Figure 2.1 : Modèle d'architecture à trois couches utilisé.

Ainsi, avec une hiérarchie décisionnelle à nombre d'étages variable, il sera possible de moduler les capacités réactives et délibératives. Ainsi, un processus décisionnel comprenant plusieurs niveaux hiérarchiques présentera forcément une délibération, une planification, fortement accrue par rapport à une prise de décision à un seul étage menant directement à un comportement, donc à capacité strictement réactive.

2.3 Architecture de contrôle générique pour robot mobile

Afin de bien situer le rôle du processus décisionnel au sein d'un robot mobile pris individuellement, traçons un schéma permettant de se représenter clairement et simplement une architecture de contrôle local.

2.3.1 Structure modulaire de l'architecture de contrôle local

La Figure 2.2 présente la structure modulaire de l'architecture de contrôle local. Cette structure sera utilisée comme modèle sous-entendu dans cet ouvrage.

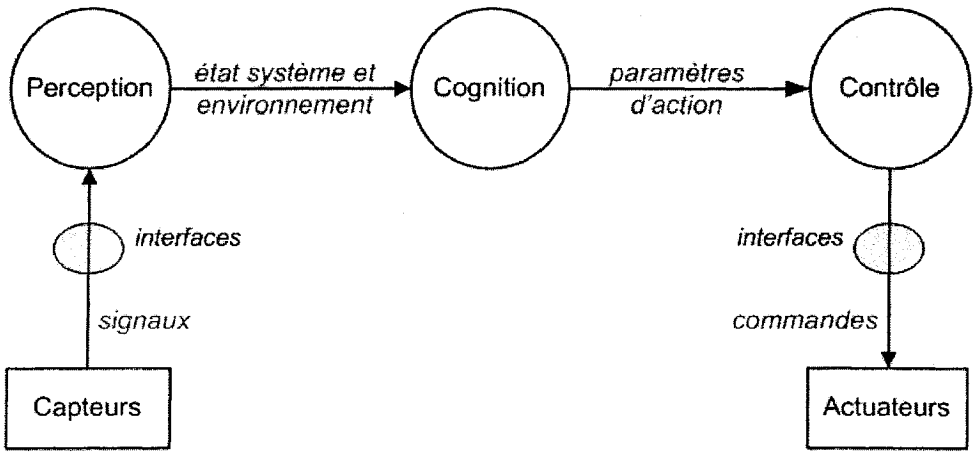


Figure 2.2 : Structure modulaire de l'architecture de contrôle local.

Module de perception

Le module (ou système) de perception d'un robot est muni de capteurs, transmettant leurs signaux via des circuits d'interfaces et pilotes adéquats au module de perception. Ce module se charge des éléments suivants :

- Modélisation du système robotisé et de son environnement
- Acquisition des informations provenant des capteurs
- Traitement et mise en forme des informations provenant des capteurs
- Mise à jour de l'état du système et de son environnement

Module de cognition

Le module qui utilisera l'information manipulée par le module de perception est le module de cognition. Le terme utilisé pour dénommer ce module est général (voir définitions en 0.3) puisque le rôle précis joué par ce module peut varier énormément en fonction de l'application du système robotisé. Ce module peut être très complexe et inclure une panoplie de procédés d'intelligence artificielle, comme il peut être très simple et n'être qu'à la limite qu'une simple fonction de l'état du système. Si nous nous référons à la définition du terme cognition donnée précédemment, il est possible d'affirmer que peu importe la complexité de ce module, son rôle au sein du logiciel est

bel et bien d'implanter toute forme d'intelligence nécessaire au fonctionnement du système robotisé. Le rôle spécifique joué par le module de cognition peut se résumer ainsi :

- Acquisition de l'état du système robotisé et de son environnement
- Traitement de l'information reçue
- Mise à jour des paramètres d'action

Le sujet principal du présent projet est bien le développement d'une architecture décisionnelle pour système multi-robots. Il va de soi donc que l'implantation distribuée d'une telle architecture se fait au sein du module de cognition de chacun des robots du système. Dans le cas d'un système à architecture hybride, le traitement d'information de ce module peut être vu comme un processus de prise de décision, de façon hiérarchisée et en fonction de l'état du système, et menant à la mise à jour des paramètres d'action.

Module de contrôle

Vient finalement le module de contrôle qui permet au système robotisé d'agir sur son environnement grâce aux actuateurs du système. Simplement, le module de contrôle joue le rôle suivant :

- Acquisition des paramètres d'action
- Traitement et mise en forme de l'information reçue
- Calcul de signaux de commande en fonction des paramètres d'action
- Transmission des signaux de commande aux actuateurs

2.3.2 Transmission d'information inter-module

Les modules principaux d'un logiciel de robot mobile viennent d'être présentés de façon simple et concise. Il faut cependant réaliser que le développement d'un tel système demande généralement d'autres éléments, comme par exemple un module de communication, et que les relations entre les différents modules ne sont pas forcément aussi simplement définies. Prenons par exemple un système robotisé utilisant une caméra motorisée pour orienter le champ de vision. Dans ce cas un lien direct entre le module de perception et le module de contrôle pourrait être nécessaire. Cependant, afin de s'assurer qu'un système effectue des actions strictement nécessaires et réfléchies, les liens directs entre les modules de perception et de contrôle sont à proscrire. Il est préférable de laisser le module de cognition transmettre toute information jugée utile au module de contrôle. Il ne s'agit pas ici de limiter ou même d'éliminer toute capacité

réactive du système, puisque la réactivité du système en sera assurée tant que le module de cognition fournit toutes les informations nécessaires à la réalisation des capacités réactives.

Par exemple, pour un robot mobile devant éviter des obstacles dans l'environnement, il est indispensable que son module de contrôle connaisse certaines informations sur les obstacles et sur l'environnement de façon à implanter un mécanisme de contrôle réactif adéquat. Le module de cognition, s'il juge qu'il est nécessaire de naviguer dans l'environnement en limitant les collisions avec les obstacles, se devra donc de transmettre au module de contrôle les informations nécessaires concernant les obstacles, informations provenant initialement du module de perception. Par contre, par délibération, le module de cognition pourrait décider de désactiver le mécanisme d'évitement d'obstacle ou encore décider qu'un obstacle en particulier devrait être traité différemment par le contrôle réactif.

Cette restriction au niveau de la transmission d'information permet donc d'assurer la possibilité de hiérarchisation du contrôle puisque le module de cognition est en mesure d'interpréter et de manipuler toute information transmise au module de contrôle. La Figure 2.3 résume la façon dont les informations doivent circuler au sein de l'architecture.

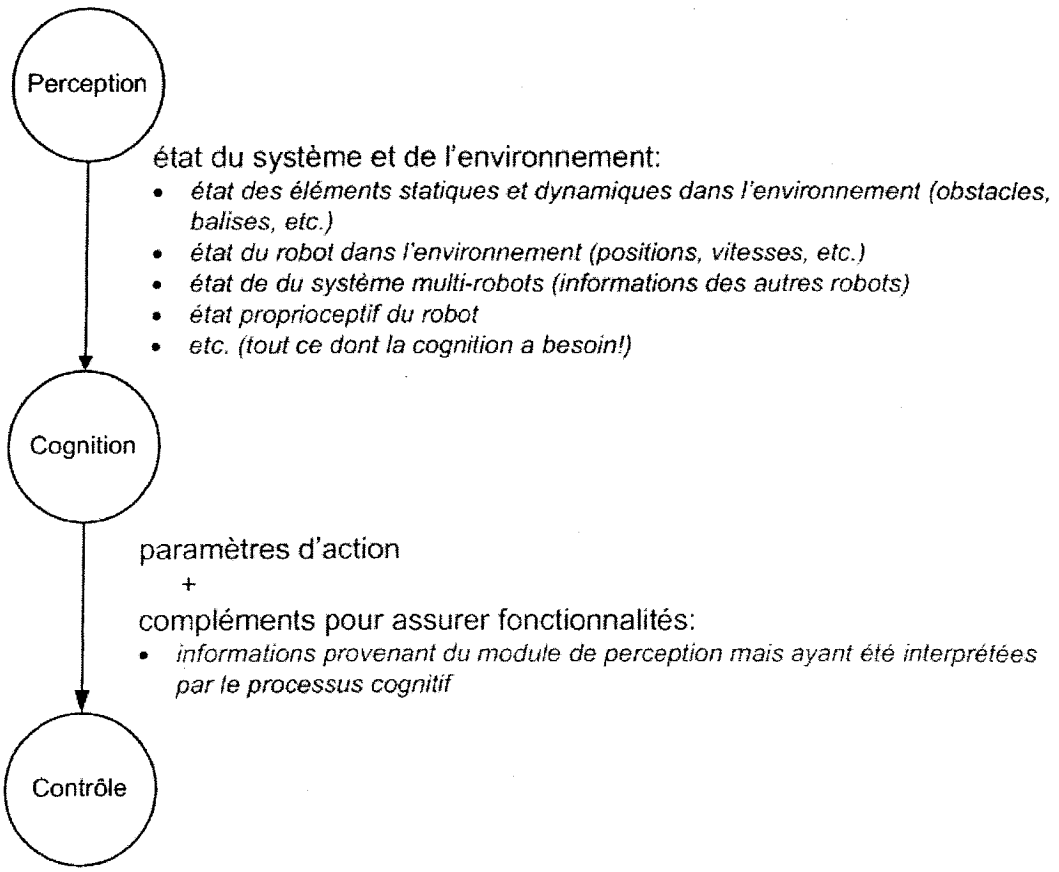


Figure 2.3 : Transmission d'information entre les différents modules de l'architecture de contrôle.

Malgré l'importance de cette restriction, certaines exceptions peuvent parfois être tolérées ou inévitables de façon à assurer les fonctionnalités requises du système.

2.3.3 Aspect temporel de l'architecture

Un système robotisé se doit d'agir et de réagir en temps réel au sein de l'environnement dans lequel il évolue. Pour ce faire, différentes implantations sont possibles pour ce qui est de la réponse temporelle du système. Par exemple, il est possible de réagir aux événements, sans boucle temporelle prédéterminée, par des méthodes d'interruption lorsqu'un événement pertinent survient. Il est aussi possible d'utiliser des boucles temporelles à période prédéterminée ou variable. Dans ce cas, dépendamment des modules de perception et de contrôle présents dans un certain système robotisé, différentes configurations sont possibles. La Figure 2.4 présente deux exemples de configurations possibles. En a), une seule boucle temporelle (« single thread ») se

charge de compléter les rôles des trois modules principaux et ce de façon séquentielle. Cette boucle sera donc exécutée de façon répétitive, en utilisant une période fixe ou variable. En b), nous retrouvons plusieurs boucles (« multi thread ») qui peuvent donc être exécutées de façon périodique, mais asynchrone et à des périodes différentes l'une de l'autre. Dans ce cas, des mécanismes comme des sémaphores, des messages et de la mémoire partagée peuvent être utilisés pour assurer un échange d'information robuste entre les différentes boucles. Le standard fourni par POSIX.4 (Gallmeister, [17]) définit les spécifications logicielles des principaux outils nécessaires à l'implantation de tels mécanismes.

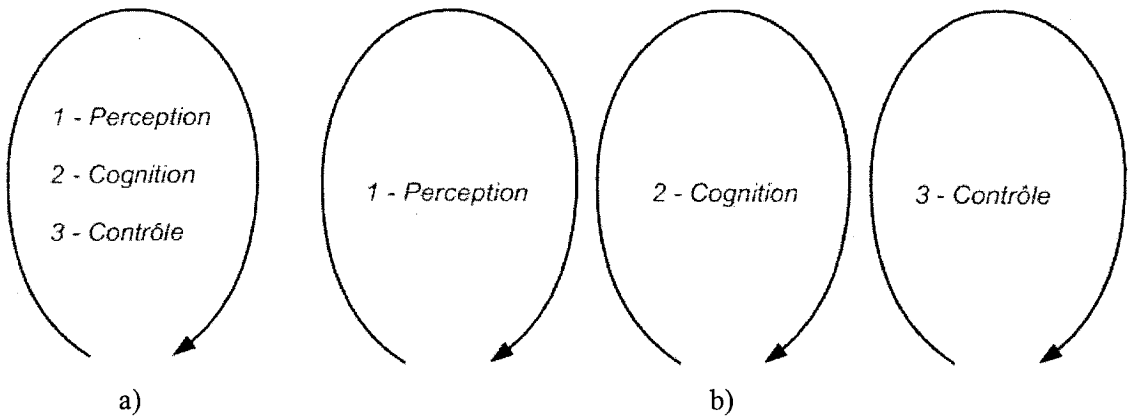


Figure 2.4 : Exemples de structures temporelles possibles pour un logiciel de système robotisé.

D'autres configurations/combinaisons d'arrangements temporels sont aussi possibles. Par exemple, le module de perception peut être divisé en plusieurs boucles s'il possède différents capteurs offrant des mises à jour d'information à intervalles différents. Les modules de cognition et de contrôle peuvent eux aussi être fractionnés en plusieurs boucles distinctes. L'utilisation d'une machine décisionnelle hiérarchique imposera très peu de restriction au niveau de la configuration temporelle du logiciel de contrôle. En effet, la seule exigence sera d'inclure la machine décisionnelle au sein d'une même boucle temporelle (ou d'une même méthode d'interruption), de façon à ce qu'elle puisse compléter son processus décisionnel en une exécution sans interruption. La machine décisionnelle étant implantée au sein du module de cognition, c'est dans ce module que la boucle associée à la machine se trouvera. La Figure 2.5 présente un exemple de structure à plusieurs boucles où le module de cognition se résume à une seule boucle constituée d'un processus de machine décisionnelle hiérarchique. Il serait aussi possible de

retrouver d'autres boucles temporelles au sein du module de cognition, par exemple pour implanter des mécanismes de contrôle de plus bas niveau, mais le processus complet de la machine décisionnelle serait encapsulé au sein d'une seule et unique boucle.

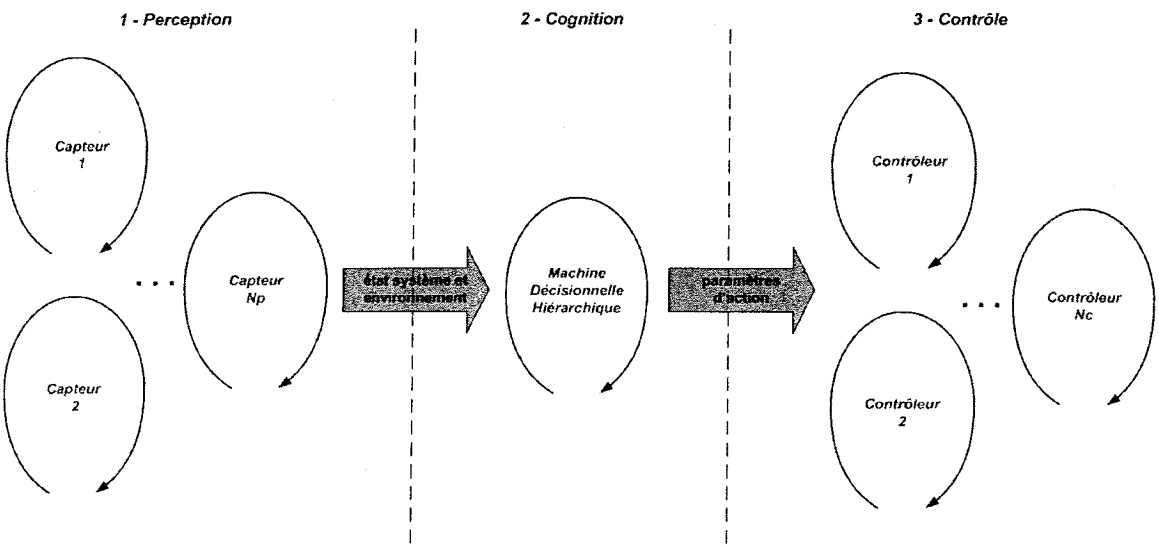


Figure 2.5 : Structure à plusieurs boucles temporelles (« multithread ») en utilisant la machine décisionnelle hiérarchique.

CHAPITRE 3 - ARCHITECTURE DÉCISIONNELLE PROPOSÉE

3.1 Machine décisionnelle hiérarchique (MDH)

Les prochaines sections présentent l'architecture développée en fonction des spécifications fonctionnelles préalablement établies et des objectifs visés de modularité, de versatilité, de représentabilité et finalement de robustesse en environnement dynamique.

Le terme « machine » est employé pour désigner l'architecture puisque cette dernière englobe tous les éléments nécessaires aux mécanismes cognitifs du système robotisé. Cette machine, située dans l'architecture à trois couches (voir Figure 2.1), permet l'implantation des deux couches de plus haut niveau, soit la couche réactive et la couche délibérative.

De façon très simple, la machine décisionnelle se veut une architecture permettant de prendre un nombre arbitraire de décisions hiérarchisées (couche délibérative) qui mèneront à divers comportements chargés de mettre à jour les paramètres d'action adéquatement (couche réactive). Puis, au plus bas de la hiérarchie, des actions sont chargées d'exécuter les consignes. Chacun des éléments de la hiérarchie, à l'exception des actions, sera représenté par un nœud. Il est préférable de donner quelques définitions avant de présenter plus en détails le concept.

Machine décisionnelle (déf.) :
Nœud principal permettant de sélectionner l'état actuel de la hiérarchie inférieure. Elle est constituée d'une liste de nœuds, ceux de l'étage inférieure, ainsi que d'un certain nombre d'étages hiérarchiques, pouvant varier en fonction de la ligne décisionnelle. La machine comprend la fonction principale permettant de lancer le processus décisionnel.

Ligne décisionnelle (déf.) :
Séquence décisionnelle, partant de la décision principale de la machine et descendant la hiérarchie décisionnelle en ne passant que par un seul nœud à chaque étage. La ligne décisionnelle passe par un nombre arbitraire de nœuds décisionnels hiérarchisés et se termine par la sélection d'un nœud comportemental.

Nœud décisionnel (déf.) :
Nœud à caractère délibératif capable de sélectionner un nœud dans l'étage hiérarchique inférieur. Il est principalement constitué d'une liste de nœuds, ceux de l'étage inférieure, ainsi que d'une fonction décisionnelle.

Nœud comportemental (déf.) :
Nœud à caractère réactif permettant l'implantation d'un comportement arbitraire. Il comporte une liste d'actions auxquelles il a accès. Son rôle est de mettre à jour les paramètres de ces actions et ensuite d'en lancer l'exécution.

Action (déf.) :
Élément permettant de compléter une action selon des paramètres provenant d'un étage supérieur. Il est constitué d'une fonction d'action qui elle seule peut accéder aux actuateurs directement ou via un intermédiaire (algorithme ou matériel de contrôle). Les actions se retrouvent toujours à l'étage le plus bas de la machine décisionnelle.

La Figure 3.1 qui suit schématise le fonctionnement général de la machine décisionnelle. On y remarque un certain nombre de décisions possibles et une ligne décisionnelle est mise en évidence. Il faut remarquer qu'un nœud décisionnel mènera toujours à un seul nœud décisionnel ou comportemental. Cependant, pour un nœud comportemental, toutes les actions lui étant associées seront exécutées. Il revient donc au comportement de bien définir les paramètres d'action pour toutes les actions qui lui sont permises.

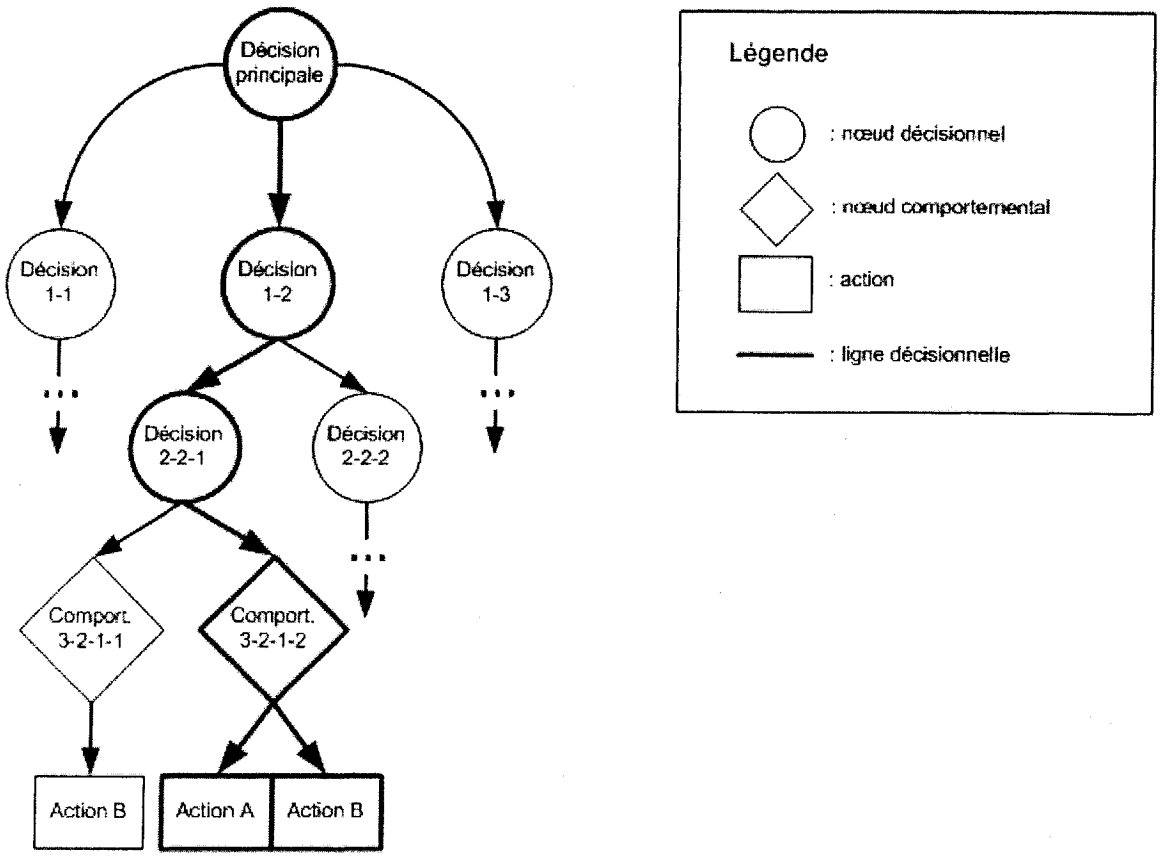


Figure 3.1 : Schéma résumant le fonctionnement de la machine décisionnelle hiérarchique.

La machine décisionnelle hiérarchique peut se représenter schématiquement sous la forme d'un arbre, et conceptuellement elle ressemble beaucoup à un concept de machines à états hiérarchisées (L'Archevêque et Dupuis, [22]). Cependant, l'utilisation du concept de machine à états doit contraindre le mécanisme décisionnel afin d'en assurer la stabilité. Une des spécifications requise de la machine décisionnelle développée (voir 2.1) est de permettre tout mécanisme décisionnel arbitraire au niveau des nœuds décisionnels. C'est pourquoi le modèle de machine à états n'est pas forcément retenu. Par ailleurs, le processus décisionnel débutant toujours par la même décision principale, par le même point d'entrée, la stabilité de la machine est assurée.

3.1.1 Mécanisme décisionnel

À chaque fois que le processus décisionnel est initié, la machine détermine de façon séquentielle la ligne décisionnelle appropriée menant jusqu'aux actions. Ainsi, chacun des étages de la

hiérarchie est parcouru, en ne passant que par un seul nœud, qui a été sélectionné par un nœud de l'étage supérieur, et chargé à son tour de sélectionner le nœud approprié.

Si le schéma de la Figure 3.1 est repris, la ligne décisionnelle mise en évidence en trait gras peut être utilisée pour illustrer le mécanisme décisionnel. Le point de départ constitue la *Décision principale* qui choisit ensuite le nœud décisionnel *Décision 1-2* qui correspond au deuxième nœud du premier étage de nœuds. Ensuite ce nœud choisit le nœud *Décision 2-2-1* qui correspond au premier nœud du second étage de nœuds, en provenance du second nœud de l'étage précédent. Et donc ensuite le nœud *Comportement 3-2-1-2* est choisi qui correspond au deuxième nœud comportemental de l'étage de nœuds 3, en provenance du nœud 1 et ainsi de suite. Ainsi, une machine décisionnelle peut comporter une quantité importante de nœuds à chaque étage et donc représenter dans l'ensemble une structure relativement imposante, mais comme tel un processus décisionnel peut se représenter de façon assez simple. Par exemple, si seul l'indice du nœud choisi à chaque étage est retenu, la ligne décisionnelle présentée précédemment peut être identifiée par les indices *2-1-2*. En identifiant ainsi chaque nœud par un indice numérique, il sera démontré plus loin toute l'utilité d'une représentation aussi simple.

Si la machine décisionnelle est utilisée dans un système multi-robots de façon distribuée, c'est-à-dire que chaque robot possède sa propre machine, les mécanismes de coopération doivent être implantés au sein de la prise de décision et des comportements. Par exemple, si un certain système possède plusieurs modes de fonctionnement et qu'idéalement un seul et même mode n'est utilisé à la fois pour l'ensemble des robots, les conditions décisionnelles doivent être prévues à cet effet. L'utilisation de l'environnement extérieur et commun à tous les robots peut alors être grandement utile. Si cette coopération intrinsèque à la machine est impossible, le seul de moyen d'assurer la coopération sera par un mécanisme de supervision (voir 3.2) nécessitant un moyen de communication explicite.

Si la supervision décisionnelle telle que présentée en 3.2 est employée, le mécanisme décisionnel peut être interrompu à un ou plusieurs étages, et la décision peut être à ce moment strictement imposée par le superviseur. Il est à noter cependant que chaque nœud peut implanter son propre mécanisme de réponse à la supervision et il est donc possible d'interpréter la demande de supervision, ajoutant ainsi aux possibilités de délibération du nœud décisionnel.

La décision comme telle n'étant pratiquement que le choix d'un nœud parmi une liste de nœuds disponibles, la fonction de décision est totalement arbitraire et donc tout type de fonction décisionnelle peut être utilisé, de la plus simple à la plus complexe. Ce mécanisme se veut donc générique et une panoplie de mécanismes évolutifs et/ou adaptifs peuvent être développés en utilisant l'architecture proposée.

3.2 Mécanisme de supervision décisionnelle

3.2.1 Problème inhérent aux systèmes à perception distribuée

L'architecture décisionnelle présentée précédemment permet à un système multi-robots de prendre des décisions de façon distribuée. C'est-à-dire qu'un robot du système sera en mesure, de façon autonome, de prendre des décisions en temps réel en se basant sur la machine décisionnelle qu'on lui a programmée. Jusqu'à présent donc, chaque machine décisionnelle implantée sur chacun des robots du système peut fonctionner de façon autonome sans nécessiter d'échange de données. Dans le cas où la perception est également distribuée, le système peut donc fonctionner sans aucune communication entre les différents robots du système.

Cependant, il peut arriver, et c'est souvent le cas avec les systèmes à perception distribuée, que certaines situations engendrent des problèmes de coordination. Ce type de problème est identifié par le terme conflit décisionnel. Le tout sera illustré par un exemple. Le schéma qui suit présente une machine décisionnelle simple pour une équipe de 2 robots joueurs de soccer. Elle est simplement constituée de deux rôles : *Offense* et *Defense*.

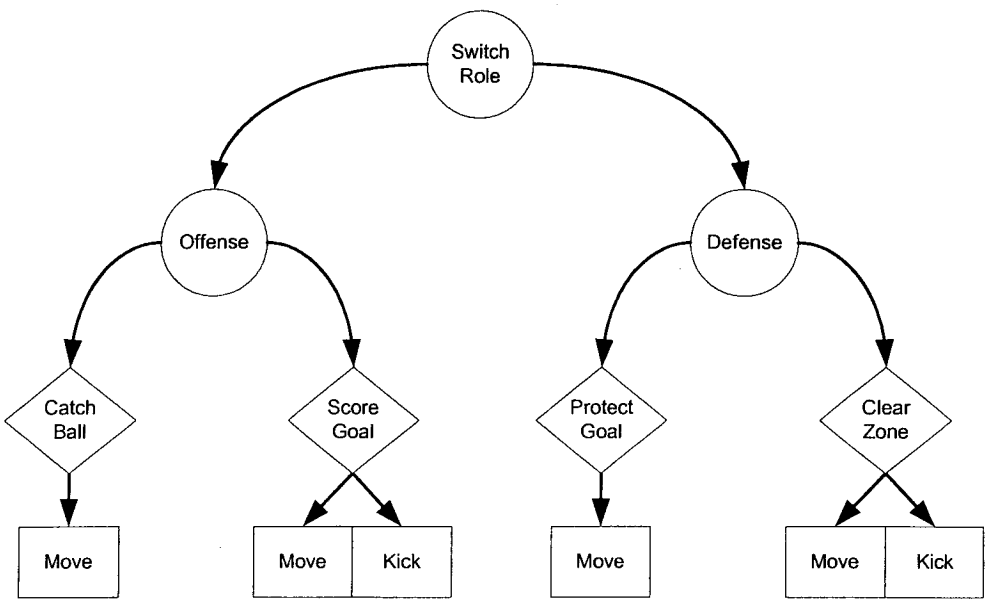


Figure 3.2 : Exemple simple de machine décisionnelle d'une équipe de 2 robots joueurs de soccer.

Cette machine pourrait être implantée de façon distribuée sur deux robots joueurs de soccer identiques et possédant leur propre perception du terrain de jeu (voir le chapitre CHAPITRE 4 - pour un exemple de tels robots). L'objectif de la machine décisionnelle présentée dans cet exemple est qu'en tout temps, un seul robot vise à jouer le rôle offensif et que l'autre vise à jouer le rôle défensif. Étant données la prise de décision distribuée, la coordination peut donc se faire à l'aide de règles implicites basées en grande partie sur l'état actuel du terrain de jeu, état déterminé par la perception distribuée sur chacun des joueurs. Voici un exemple simple basé sur la position relative au but adverse :

⇨ **SI** je suis le joueur le plus près du but adverse

⇨ **ALORS** je prends le rôle offensif

⇨ **SINON**

⇨ **ALORS** je prends le rôle défensif

Cette règle décisionnelle est bien simple et peut fonctionner correctement dans le cas où la condition concernant la distance par rapport au but adverse est évidente. Par exemple la figure

suivante illustre une situation de jeu où l'attribution des rôles se fera sans aucun problème. Cette situation devrait mener aux lignes décisionnelles présentées à la Figure 3.4.

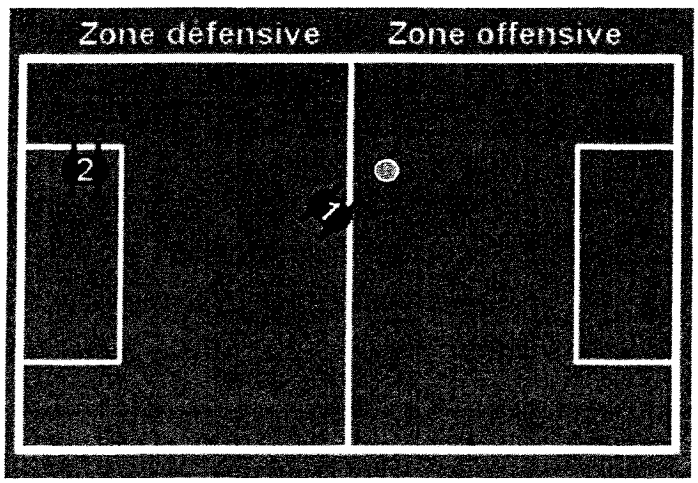


Figure 3.3 : Situation de jeu, équipe de 2 robots joueurs de soccer (attribution de rôles correcte).

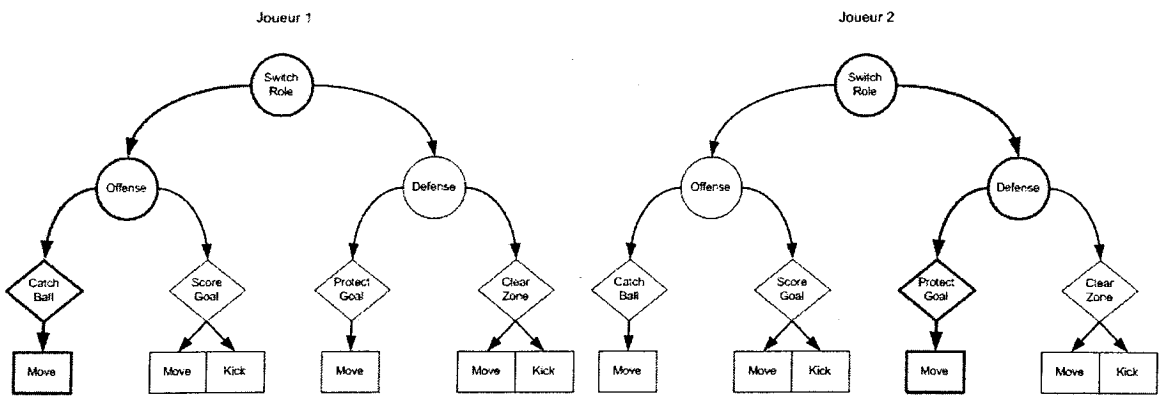


Figure 3.4 : Attribution des rôles correcte, équipe de 2 robots joueurs de soccer.

Par contre, il peut facilement survenir des situations susceptibles de générer des conflits décisionnels. En observant maintenant la situation de jeu présentée à la Figure 3.5, il faut remarquer que les distances respectives des deux joueurs par rapport au but adverse sont similaires. De plus, même si ces distances ne sont pas en fait exactement égales, les erreurs de perception associées aux systèmes à perception distribuée peuvent engendrer des erreurs décisionnelles. Une telle situation pourrait donc engendrer un conflit décisionnel. Par exemple, à

la Figure 3.6, on peut constater un conflit décisionnel puisque les deux robots choisissent le rôle offensif.

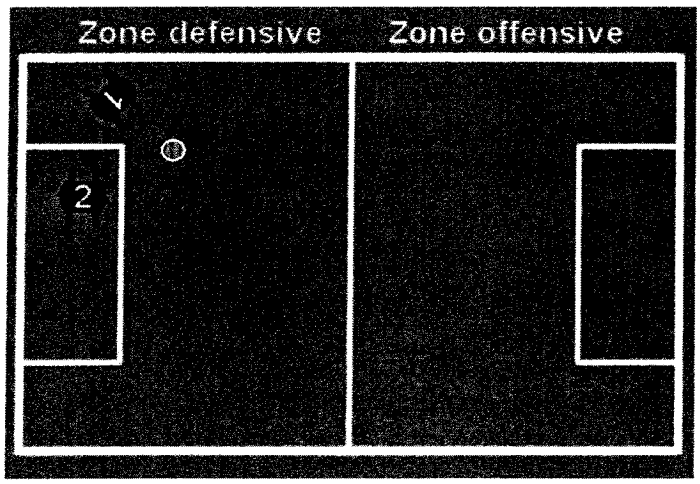


Figure 3.5 : Situation de jeu, équipe de 2 robots joueurs de soccer (possibilité de conflit décisionnel).

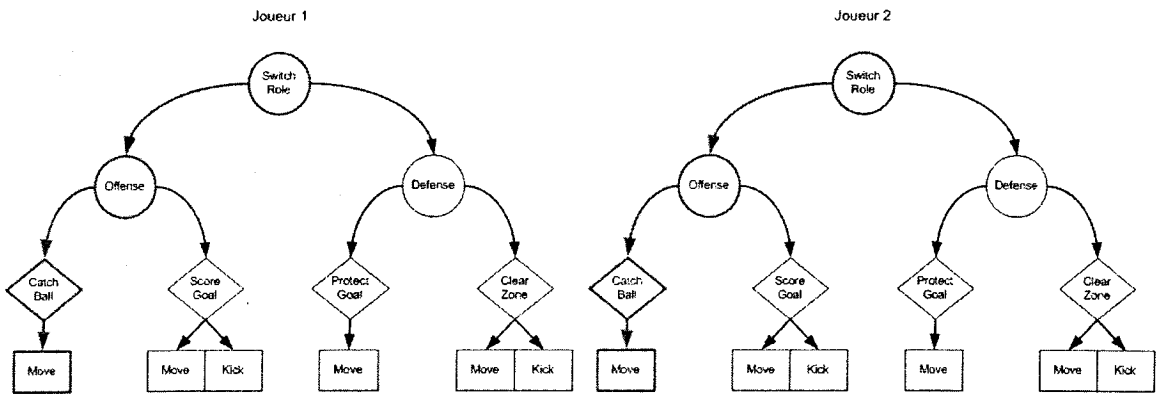


Figure 3.6 : Conflit décisionnel, équipe de 2 robots joueurs de soccer.

3.2.2 Supervision décisionnelle centralisée

L'exemple de l'équipe de 2 robots joueurs de soccer démontre la possibilité de conflits décisionnels pour un système multi-robot coopératif à prise de décision distribuée. Même s'il y coopération intrinsèque au sein de la machine décisionnelle, les conflits décisionnels sont dans certaines situations inévitables. Les risques de conflits décisionnels sont encore plus grands lorsque ce système utilise une perception distribuée qui est imprécise. Il est donc nécessaire, si le but est d'obtenir un système coopératif qui fonctionne de façon autonome et robuste, de prévoir certains mécanismes permettant d'éliminer les conflits décisionnels. Plusieurs méthodes de

coordination inter-robot ont déjà été développées et sont assez répandues, par exemple des méthodes de négociation entre robots (Emery et coll., [16]).

Dans le cas du présent projet, il est souhaitable de limiter les besoins en communication ainsi que de limiter le plus possible l'intervention au niveau de la prise de décision distribuée développée jusqu'à présent. Nous cherchons de plus à bénéficier des avantages de deux approches bien distinctes : l'approche à prise de décision distribuée et celle avec prise de décision centralisée. La prise de décision distribuée permet une vitesse de réponse et d'exécution intéressante et la possibilité de fonctionner sans communication. C'est cette approche qui est présentée et développée jusqu'à présent. La prise de décision centralisée permet elle d'éliminer tous les conflits décisionnels puisqu'une seule entité est en mesure de prendre les décisions et doit les communiquer aux autres entités du système.

L'idée du mécanisme développé ici est de permettre, lorsque la situation le dicte, à une entité de superviser la prise de décision des différents robots du système. Cette entité, qui sera désignée par le terme « superviseur », peut se développer sous différentes formes : un processus autonome qui décide par lui-même des décisions à imposer, ou encore un opérateur humain muni d'une interface adéquate.

Grâce à l'architecture décisionnelle développée, un mécanisme de supervision décisionnelle centralisée est facile à implanter et la supervision peut être faite sur toute la hiérarchie décisionnelle. Cette facilité vient du fait que chaque nœud (décisionnel et comportemental) d'une machine donnée peut être identifié individuellement et facilement retrouvé au sein de la hiérarchie.

La machine décisionnelle de la Figure 3.7 servira d'exemple pour expliquer le fonctionnement de la supervision décisionnelle centralisée. Le schéma présente deux lignes décisionnelles distinctes. Si l'identité numérique (valeur numérique *ID* sur le schéma) est utilisée pour adresser chacun des nœuds de la hiérarchie, une ligne décisionnelle peut être représentée par une suite de nombres entiers. Par exemple, la ligne décisionnelle à gauche peut se représenter par la suite [0,2] et la ligne décisionnelle à droite peut se représenter par [2,0].

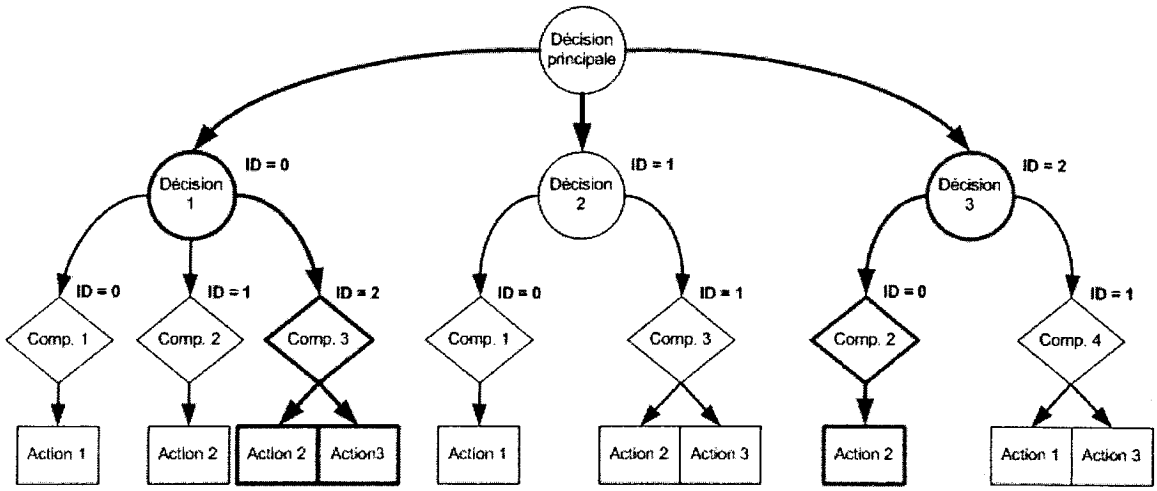


Figure 3.7 : Exemple de machine décisionnelle pour illustrer la supervision décisionnelle.

La supervision décisionnelle a donc comme objectif d'imposer lorsque nécessaire un nombre arbitraire de décisions allant jusqu'à la ligne décisionnelle complète. Il n'est pas obligatoire d'imposer la décision sur toute la hiérarchie, par exemple dans la machine illustrée à la Figure 3.7, la supervision pourrait se faire seulement au niveau du choix entre *Decision 1*, *2* ou *3*. Et si l'exemple de l'équipe de 2 robots joueurs de soccer est considéré, le conflit décisionnel de la Figure 3.6 peut être évité en imposant deux rôles distincts à chacun des robots.

Lorsqu'un robot reçoit une décision supervisée sous la forme d'une suite d'identités de nœuds, il a la responsabilité de la respecter, mais il garde toute sa liberté d'action au niveau du comportement actif. En effet, la supervision décisionnelle s'arrête au choix du comportement et le superviseur n'est pas en mesure d'influencer le comportement sélectionné.

Pour ce qui est du mécanisme de supervision comme tel, son implantation est dictée par l'application. Certaines applications vont nécessiter un superviseur humain, tandis que d'autres vont demander à ce que le système multi-robots soit entièrement autonome et donc que le superviseur le soit également. La représentation simple de la machine décisionnelle devrait permettre tout type de mécanisme de supervision.

3.2.3 Représentation des informations décisionnelles

L'utilisation d'identificateurs numériques pour la distinction des nœuds de la machine décisionnelle est essentielle à une représentation simple des informations décisionnelles. Deux

contraintes doivent être respectées pour assurer le bon fonctionnement du mécanisme de supervision :

- L'identificateur est une valeur numérique entière supérieure ou égale à zéro
- Une valeur unique est attribuée à chaque nœud d'une liste de nœuds donnée

Afin de simplifier également la représentation du système multi-robots, il est préférable d'identifier chacun des robots du système par une suite de nombre entiers incrémentée de 1 et débutant par 0.

En utilisant cette notation, des structures de données peuvent définies pour l'implantation de la supervision décisionnelle. Premièrement, la ligne décisionnelle peut être représentée sous la forme d'un vecteur décisionnel. Ensuite, l'ensemble des vecteurs décisionnels des robots d'un système multi-robot peuvent être regroupés sous la forme d'une matrice décisionnelle. De façon similaire, la supervision décisionnelle pour l'ensemble d'un système multi-robots peut être représentée par une matrice de supervision et la supervision de chacun des robots peut se faire par un vecteur de supervision. En considérant un système multi-robots composé de N_R robots utilisant chacun une machine décisionnelle comportant N_D hiérarchies décisionnelles (ce nombre peut varier dynamiquement), ces structures de données peuvent se représenter ainsi :

Vecteur décisionnel (déf.) :

Vecteur ligne de dimension N_D représentant la ligne décisionnelle du robot *i*.

$$V_{D_i} = \begin{bmatrix} D_{i-1} & D_{i-2} & \dots & D_{i-N_D} \end{bmatrix}$$

Matrice décisionnelle (déf.) :

Matrice de N_R lignes par N_D colonnes où les lignes représentent les lignes décisionnelles de chacun des robots du système multi-robots.

$$M_D = \begin{bmatrix} D_{1-1} & D_{1-2} & \cdots & D_{1-N_D} \\ D_{2-1} & D_{2-2} & \cdots & D_{2-N_D} \\ \vdots & \vdots & \vdots & \vdots \\ D_{N_R-1} & D_{N_R-1} & \cdots & D_{N_R-N_D} \end{bmatrix}$$

Matrice de supervision (déf.) :

Matrice de N_R lignes par N_D colonnes où les lignes représentent les vecteurs de supervision pour chacun des robots du système multi-robots.

$$M_S = \begin{bmatrix} S_{1-1} & S_{1-2} & \cdots & S_{1-N_D} \\ S_{2-1} & S_{2-2} & \cdots & S_{2-N_D} \\ \vdots & \vdots & \vdots & \vdots \\ S_{N_R-1} & S_{N_R-1} & \cdots & S_{N_R-N_D} \end{bmatrix}$$

Vecteur de supervision (déf.) :

Vecteur ligne de dimension N_D représentant la supervision décisionnelle du robot *i*.

$$V_{S_i} = \begin{bmatrix} S_{i-1} & S_{i-2} & \cdots & S_{i-N_D} \end{bmatrix}$$

Il est important de noter que le vecteur de supervision, au sein d'un robot, ne sera jamais mis à jour par ce dernier. Il demeurera constant tant qu'il ne sera pas mis à jour par le superviseur.

Ces structures de données faciliteront grandement le développement de mécanismes de supervision par leur représentation vectorielle simple.

3.2.4 Architecture de communication (centralisation)

Le concept de supervision sur un systèmes multi-robots requiert une certaine forme de communication explicite. Dépendamment du système concernée, différentes architectures de communication sont possibles de façon à permettre l'utilisation de machines décisionnelles distribuées. La Figure 3.8 présente quatre architectures qui reflètent des configurations couramment utilisées sur des systèmes expérimentaux.

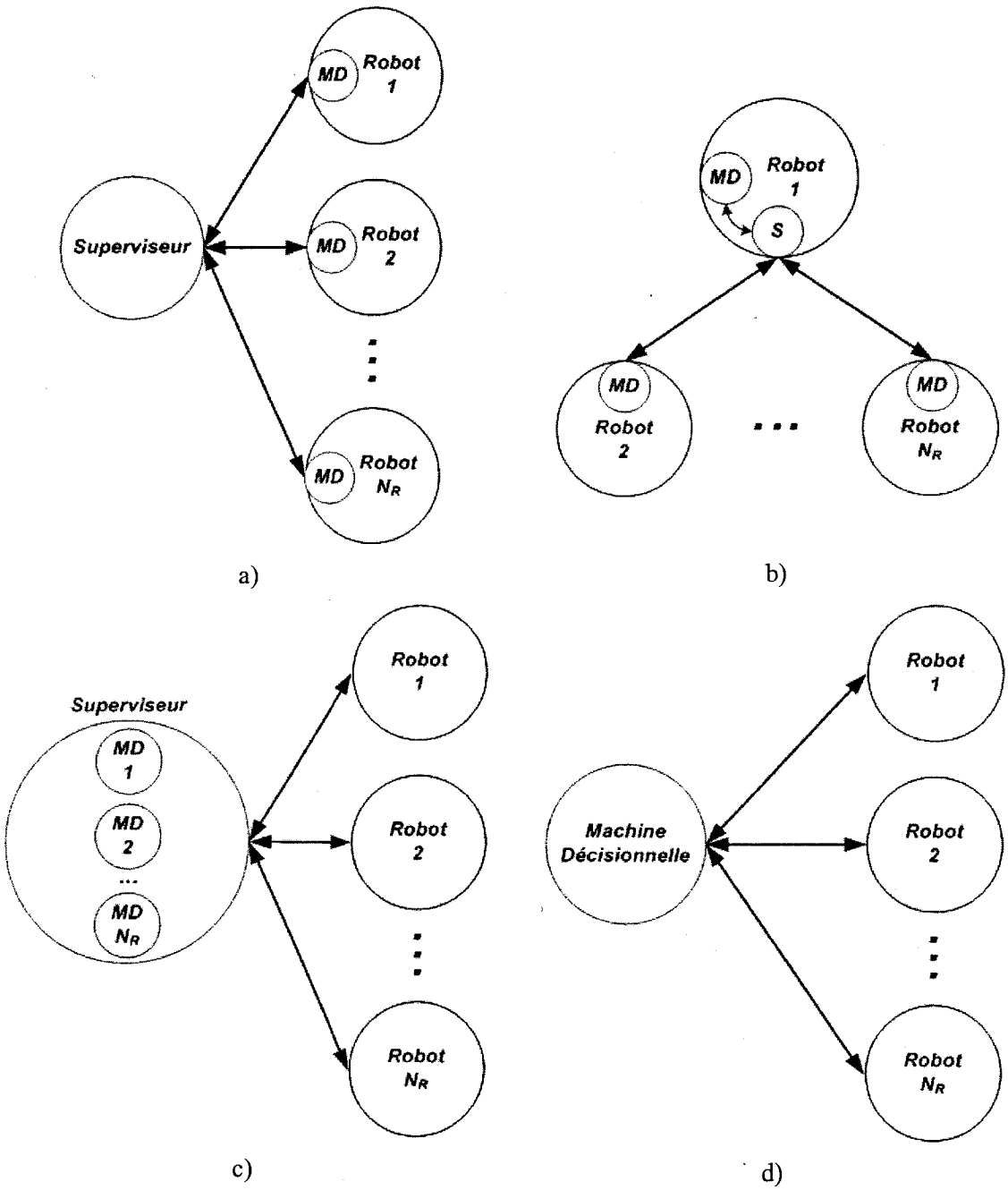


Figure 3.8 : Architectures de communication possibles pour systèmes multi-robots avec machines décisionnelles et superviseur.

En a) l'architecture est distribuée avec une machine décisionnelle par robot ainsi qu'un superviseur étant un module externe à chacun des robots qui doivent communiquer avec ce dernier. En b), l'architecture est similaire mais le superviseur se retrouve sur un des robots. Le

robot possédant le robot-superviseur peut bénéficier de délais de communication beaucoup plus courts mais globalement le fonctionnement du système peut être le même qu'en a), à moins que le robot-superviseur ne bénéficie d'informations supplémentaires disponibles au sein du superviseur. Cette architecture a l'avantage de ne pas demander un ordinateur supplémentaire mais le superviseur doit être un processus autonome (interface homme-machine plus difficile) et les ressources doivent être suffisantes au sein de l'ordinateur du robot-superviseur. En c), l'architecture est beaucoup plus centralisée où les machines décisionnelles se retrouvent sur le même ordinateur que le superviseur. En d), l'architecture est tout aussi centralisée et il est possible dans ce cas de n'utiliser qu'une seule machine décisionnelle pour contrôler l'ensemble des robots du système. Chaque robot peut ainsi comporter son propre ensemble d'actions au sein de la machine. La configuration en d) peut donc très bien remplacer celle en c) qui est plus complexe. Les options c) et d) sont intéressantes pour certains systèmes multi-robot munis de robots simples répondant à des commandes de bas niveau provenant d'un ordinateur capable lui d'intelligence de plus haut niveau.

3.3 Modélisation informatique

Voici un résumé des principales composantes et caractéristiques des éléments logiciels développés dans le cadre de ce projet. L'implantation est faite en langage C++ et donc en programmation par objet. Les classes permettant d'implanter la machine décisionnelle hiérarchique sont donc présentées. Les fichiers concernés sont présentés par leur fichier d'entête de classe. Cela n'est peut-être pas la façon la plus élégante de présenter l'implantation, mais les classes étant relativement petites et simples, cela est très efficace comme présentation. La documentation complète du code a par ailleurs été générée à partir de Doxygen (van Heesch, [19]).

Comme il a été présenté préalablement, l'architecture décisionnelle est principalement composée de 4 éléments : la machine décisionnelle comme tel, les nœuds décisionnels, les nœuds comportementaux ainsi que les actions.

3.3.1 Modélisation des nœuds

3.3.1.1 Nœud de base (*HDM_node*)

Cette classe constitue la classe de base pour tous les éléments de la machine décisionnelle, sauf les actions (classe *Action*) qui sont différentes. Ainsi, le nœud décisionnel (classe *Decision_node*), le nœud comportemental (classe *Behavior_node*) et l'arbre décisionnel (classe *HDM_tree*) sont tous modélisés par des classes qui héritent de celle du nœud de base. Le nœud de base contient donc les éléments essentiels à ces classes.

Notons que cette classe est une classe virtuelle pure puisqu'elle ne sera jamais utilisée directement. Elle sera toujours utilisée sous une forme dérivée.

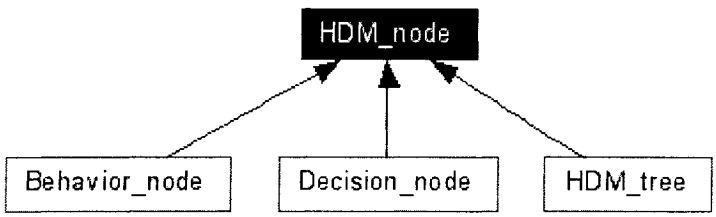


Figure 3.9 : Diagramme d'héritage de la classe *HDM_node* (généré par Doxygen, van Heesch, [18]).

```
#ifndef HDM_NODE_H
#define HDM_NODE_H

#include "HDM.h"
#include "Creator.h"

enum NODE_TYPE {DECISION, BEHAVIOR, TREE};

class HDM_node
{
private:
    void get_pointers();
    int pointers_ok;

protected:
    char *name;
    int id;
    int node_type;

    Creator * creator;
    BrainController * braincontroller;
    cPerception * perception;
    DataAcquisition * data;

    struct Robot_info rbt_info;
    struct Field_info field_info;
    double period;
```

```
    Configuration config;

public:
    HDM_node();

    ~HDM_node();

    virtual void behavior() = 0;

    virtual HDM_node *decision(int ID=-1) = 0;

    virtual void update_info();

    void set_name(char *name_) {name = name_;};

    void set_id(int id_) {id = id_;};

    char *get_name(){return name;};

    int get_id(){return id;};

    int get_type(){return node_type;};

    virtual void set_field_info(const struct Field_info fi){field_info = fi;};

    virtual void set_robot_info(const struct Robot_info rbt);

};
#endif
```

Figure 3.10 : Fichier d'entête de la classe *HDM_node*.

On peut remarquer quelques éléments particulièrement importants dans cette classe. Premièrement l'identité du nœud peut être donnée par une chaîne de caractères (**name*) et un identificateur numérique (*id*) utilisée entre autre pour la surcharge décisionnelle. Le nœud est également muni d'un *type* qui peut prendre les valeurs *DECISION*, *BEHAVIOR* et *TREE*. Ensuite le nœud a accès aux modules de perception et de cognition de l'architecture (**braincontroller* et **perception* dans l'implantation présentée). La machine doit modéliser et connaître son état (**rbt_info*) ainsi que l'état de l'environnement (**field_info*). Ces structures de données sont assez centrales au fonctionnement de la machine puisqu'elles doivent contenir tout ce qu'un étage hiérarchique peut nécessiter pour implanter son mécanisme, décisionnel ou comportemental. Finalement, un groupe de méthodes viennent assurer les fonctionnalités requises par les nœuds de la machine.

3.3.1.2 Nœud décisionnel (*Decision_node*)

Le nœud décisionnel hérite donc des attributs de la classe *HDM_node* et possède les éléments nécessaires pour prendre une décision à savoir le choix d'un nœud approprié parmi une liste. La figure qui suit fournit l'entête du fichier de la classe *Decision_node*.

```
#ifndef DECISION_NODE_H
#define DECISION_NODE_H

#include "HDM_node.h"

class Decision_node: public HDM_node
{
protected:
    Nlist node_list;
    HDM_node *current;
    HDM_node *previous;

public:
    Decision_node();
    ~Decision_node();

    virtual HDM_node *decision(int ID=-1) = 0;

    void behavior() {};

    virtual void set_field_info(const struct Field_info fi);
};
#endif
```

Figure 3.11 : Fichier d'entête de la classe *Decision_node*.

Le nœud décisionnel est principalement composé d'une liste de nœuds (*node_list*), ceux de l'étage inférieur, de pointeurs vers les nœuds courant et précédent (**current* et **previous*) ainsi que d'une méthode de décision *decision (int ID=-1)*. Cette méthode reçoit en argument la valeur du nœud de supervision, la valeur par défaut (-1) étant la valeur lorsqu'il n'y a pas de supervision (tous les nœuds ont un *id* supérieur ou égal à zéro). La valeur de retour de cette méthode est un pointeur vers le nœud choisi parmi la *node_list*. Cette classe est également virtuelle pure puisque que comme tel elle n'implémente aucun processus cognitif, une classe l'utilisera, par héritage, de façon à implanter un nœud avec processus délibératif.

3.3.1.3 Nœud comportemental (*Behavior_node*)

Le nœud comportemental est implanté de façon similaire au nœud décisionnel, c'est-à-dire que sa classe hérite du nœud de base (*HDM_node*) et que c'est une classe virtuelle pure qui sera utilisée par héritage pour implémenter divers comportements. Voici le fichier d'entête de la classe *Behavior_node*.

```
#ifndef BEHAVIOR_NODE_H
#define BEHAVIOR_NODE_H

#include "HDM_node.h"
#include "Action.h"

class Behavior_node: public HDM_node
{
protected:
    Nlist action_list;
    struct Action_parameters action_parameters;

public:
    Behavior_node();
    ~Behavior_node();

    void behavior();

    virtual void update_params() = 0;

    void execute_actions();

    HDM_node *decision(int ID=-1) {return NULL;};

    virtual void set_field_info(const struct Field_info fi);
};
#endif
```

Figure 3.12 : Fichier d'entête de la classe *Behavior_node*.

Ce nœud comprend principalement une liste d'actions possibles (*action_list*), un ensemble de paramètres d'action (*action_parameters*) ainsi que trois méthodes permettant de mettre à jour ces paramètres et d'exécuter les actions correspondantes. Le tout se fait via la méthode *behavior()* dont l'implantation toute simple est présentée ci-bas.

```
void Behavior_node::behavior()
{
    update_params();
    execute_actions();
}
```

Figure 3.13 : Implantation de la méthode *Behavior_node::behavior()*.

La méthode *execute_actions()* se charge de parcourir la liste *action_list* et d'exécuter toutes les actions s'y trouvant en utilisant les paramètres d'action mis à jour. Cette méthode est implantée au sein de la classe *Behavior_node*. Ainsi, la seule méthode qui doit être surchargée par la classe dérivée, le comportement implémenté, est la méthode *update_params()* qui n'a qu'à mettre à jour les paramètres d'action, le seul rôle joué par les divers comportements.

3.3.1.4 Action (*Action*)

La classe *Action* contient les éléments permettant d'exécuter les différentes actions du système concerné. Cette classe ne dérive pas de *HDM_node* comme les deux précédentes puisqu'elle ne constitue pas un nœud comme tel. Elle présente tout de même une structure similaire. La classe *Action* est également une classe virtuelle pure. La figure qui suit présente son fichier d'entête.

```
#ifndef ACTION_H
#define ACTION_H

#include "HDM.h"
#include "Creator.h"

class Action
{
private:
    void get_pointers();
    int pointers_ok;

protected:
    Creator * creator;
    Controller * ctrl;
    MotionControl * motion;
    DataAcquisition * data;
    struct Robot_info rbt_info;
    struct Field_info field_info;
    Configuration config;

public:
    Action();
    virtual ~Action(){};

    virtual void action(Action_parameters parameters) = 0;

    virtual void update_info();

    virtual void set_field_info(const struct Field_info fi){field_info = fi;};

    virtual void set_robot_info(const struct Robot_info rbt){rbt_info = rbt;};
};
#endif
```

Figure 3.14 : Fichier d'entête de la classe *Action*.

On peut remarquer qu'elle est la seule classe à avoir accès à la première couche de l'architecture de contrôle (voir Figure 2.1), soit la couche de commande. Dans le cas présent elle accède à cette couche par les pointeurs **ctrl* et **motion*. Tout comme les nœuds, les actions possèdent les structures de données donnant l'état du robot (**rbt_info*) ainsi que l'état de l'environnement (**field_info*). La méthode qui permet d'exécuter les actions est *action(Action_parameters*

parameters) qui reçoit les paramètres d'action à exécuter. Cette méthode est la seule à surcharger du côté de la classe dérivée.

3.3.2 Modélisation de la machine décisionnelle (*HDM_tree*)

La machine décisionnelle est implantée sous l'appellation *HDM_tree* en raison de sa représentation sous forme d'arbre. La classe *HDM_tree* hérite également de la classe *HDM_node* puisqu'elle partage un ensemble d'attributs communs. Elle est une classe virtuelle pure, donc toute machine décisionnelle développée se devra d'hériter de cette classe de base.

```
#ifndef HDM_TREE_H
#define HDM_TREE_H

#include <malloc.h>
#include "HDM.h"
#include "HDM_node.h"

class HDM_tree : public HDM_node
{
private:

protected:
    int nb_layers;
    Nlist layer_names;
    Nlist node_list;
    HDM_node * maindecision;
    HDM_node * currentnode;
    int *current_decision;
    int *supervised_decision;

public:
    HDM_tree();
    ~HDM_tree();

    virtual int build() = 0;

    virtual void process();

    void behavior() {};

    HDM_node *decision(int ID=-1) {return NULL;};

    void set_maindecision(HDM_node *md) {maindecision = md;};

    void set_maindecision(const int ID) {reset_decision(); maindecision =
(HDM_node *)node_list.GetData(ID);};

    void set_decision(int *d) {supervised_decision = d;};

    int * get_decision() {return current_decision;};

    void reset_decision();

    virtual void set_field_info(const struct Field_info fi);
```

```
    virtual void set_data_output(const int data_output_){data_output =
data_output_;};

    virtual int get_nb_layers(){return nb_layers;};
};
#endif
```

Figure 3.15 : Fichier d'entête de la classe *HDM_tree.*

Cette classe possède plusieurs éléments importants, à commencer par les principales structures de données de l'architecture décisionnelle : le nombre d'étages de la hiérarchie décisionnelle (*nb_layers*), les noms des différents étages de la hiérarchie décisionnelle (*layer_names*), une liste de nœuds qui sont les nœuds de l'étage au-dessous (*node_list*), le vecteur décisionnel (**current_decision*) et le vecteur de supervision (**supervised_decision*). Le nombre n'étages *nb_layers* correspond en fait au nombre maximal d'étages possible (la plus longue ligne décisionnelle possible), mais le nombre d'étages en cours d'exécution peut varier dynamiquement. La classe possède également quelques méthodes particulières comme la méthode du processus décisionnel comme tel (*process()*) qui sera la méthode appelée pour lancer le processus de la machine décisionnelle. Cette méthode est implantée au sein de la classe *HDM_tree* et elle n'a donc pas besoin d'être surchargée pour chaque machine décisionnelle développée (elle peut l'être si nécessaire). La seule méthode à surcharger est *build()* qui permet de bâtir comme tel la hiérarchie décisionnelle (définir le nombre d'étages, la liste de nœuds et etc.), c'est tout! Pour permettre la supervision décisionnelle, la méthode *set_decision(int *d)* permet simplement de mettre à jour le vecteur de supervision reçu en paramètre.

3.3.3 Implantation du mécanisme de supervision

Le superviseur comme tel sera toujours un module externe à la machine décisionnelle. Ainsi, au sein de la machine décisionnelle, il y a très peu d'éléments à développer afin de permettre la supervision décisionnelle. Il faut simplement voir à ce qu'un superviseur puisse envoyer à une machine donnée, à un robot, le vecteur de supervision. Reste ensuite à la machine à interpréter ce vecteur de supervision. Généralement il y aura tout simplement décision imposée mais parfois par délibération la machine pourrait réagir autrement.

La machine permet la réception du vecteur de supervision par la méthode *HDM_tree::set_decision(int *d)*. Ensuite, au sein de la méthode *HDM_tree::process()*, tout est

prévu pour l'interprétation du vecteur de supervision. La figure qui suit présente le code source de cette méthode.

```
Nlist HDM_tree::process()
{
    layer_names.Front();

    int i;

    // pour verifier jusqu'a quel niveau la decision se rend
    for(i=0;i<nb_layers;i++)
        current_decision[i] = -1;

    // si decision supervisee pour la decision principale
    if(supervised_decision[0] != -1)
        maindecision = (HDM_node *)node_list.GetData(supervised_decision[0]);
    currentnode = maindecision;
    current_decision[0] = currentnode->get_id();

    int done = 0;
    i = 1;
    while(currentnode->get_type() != BEHAVIOR)
    {
        currentnode->set_robot_info(rbt_info);
        currentnode = currentnode->decision(supervised_decision[i]);
        current_decision[i] = currentnode->get_id();

        layer_names.Next();
        i++;
    }
    currentnode->set_robot_info(rbt_info);
    currentnode->behavior();
};
```

Figure 3.16 : Fichier *process()* de la classe *HDM_tree.*

On remarque que s'il y a supervision pour la décision principale, par défaut (via la classe de base), la méthode ne fait qu'imposer la décision et donc choisir le nœud correspondant :

```
maindecision = (HDM_node *)node_list.GetData(supervised_decision[0]);
```

Ensuite, lors du parcours de la hiérarchie décisionnelle dans la boucle « while », la décision supervisée est bien transmise à chacun des nœuds de la ligne décisionnelle :

```
currentnode = currentnode->decision(supervised_decision[i]);
```

Afin d'assurer la transmission du vecteur décisionnel au superviseur, cette méthode se charge également de la mise à jour du vecteur :

```
current_decision[i] = currentnode->get_id();
```

Comme il a été mentionné en 3.2.4, une communication entre la machine et le superviseur est forcément requise. Aucun mécanisme de communication n'est cependant prévu pour l'instant. La

raison étant entre autre que cela permet de conserver l'aspect générique de la machine, mais surtout que la machine s'intègre au sein d'une architecture de contrôle qui contient généralement tous les éléments nécessaires au bon fonctionnement du système. Par exemple, pour un système multi-robots coopératif, rares sont les implantations d'architectures de contrôle qui ne prévoient pas certains mécanismes de communication. Par exemple, pour le développement expérimental présenté au chapitre suivant, l'architecture utilise déjà et ce plusieurs usages différents des communications par « sockets » TCP/IP implantés sous formes de clients et serveurs. Ainsi, le mécanisme de communication pour la supervision décisionnelle ne sera à ce moment que le simple ajout d'un service permettant d'échanger les informations nécessaires (vecteur de décision et vecteur de supervision).

CHAPITRE 4 - IMPLANTATION SUR UN SYSTÈME MULTI-ROBOTS EXPÉRIMENTAL

4.1 Objectifs

Le développement d'une équipe de robots joueurs de soccer devrait donc permettre de valider en quelque sorte le concept d'architecture décisionnelle proposé. Comme il a été mentionné précédemment, le soccer robotisé constitue une plateforme très intéressante pour le développement de systèmes multi-robots autonomes. L'environnement de jeu comprend une dynamique rapide et imprévisible et le sport comme tel permet à plusieurs équipes de se rencontrer lors d'événements internationaux et d'échanger lors des compétitions et conférences. Un autre aspect intéressant de la « robotique sportive » est l'accès à des mesures de performance grâce aux statistiques du jeu (buts marqués, temps de possession, nombre de tirs, etc.). Selon Peter Stone (Stone, [39], p. 198), le soccer robotisé permet d'adresser l'ensemble des 17 problèmes qu'il juge propres aux systèmes multi-agents.

En plus des objectifs propres au développement d'une architecture décisionnelle, il y a plusieurs objectifs reliés au développement d'une équipe de robots joueurs de soccer comme tel. En utilisant la machine décisionnelle développée, il est prévu de tester différents patrons de jeu d'attaque et de défense de façon à sélectionner, de façon arbitraire ou encore par mécanisme automatique, afin d'obtenir une équipe dont les performances s'améliorent graduellement. La machine décisionnelle doit également permettre de simplifier et d'accélérer le développement de toutes les fonctionnalités nécessaires pour respecter les règles imposées par la « Middle Size Robot League » de la *RoboCup* (MSL Technical Committee, [26]). Finalement, le développement d'une machine décisionnelle au sein d'un système multi-robots fonctionnel doit permettre à une équipe pluridisciplinaire d'étudiants (Robofoot ÉPM, [36]) de participer à plusieurs compétitions internationales régies par la fédération *RoboCup* (RoboCup Federation, [34]).

4.2 Présentation de la plateforme d'essai

De nombreuses composantes sont nécessaires au fonctionnement du système de jeu de soccer robotisé dans son ensemble. On compte parmi ces éléments une plateforme mécatronique, les

modules électroniques essentiels, un logiciel de contrôle temps-réel ainsi qu'un simulateur. Tous ces éléments permettent de développer un système multi-robots coopératif. Le système développé dans le cadre de ce projet compte 6 robots physiquement homogènes. L'image qui suit présente l'aspect physique de ces robots et les lignes qui suivent présentent les divers éléments développés dans le cadre de ce projet.

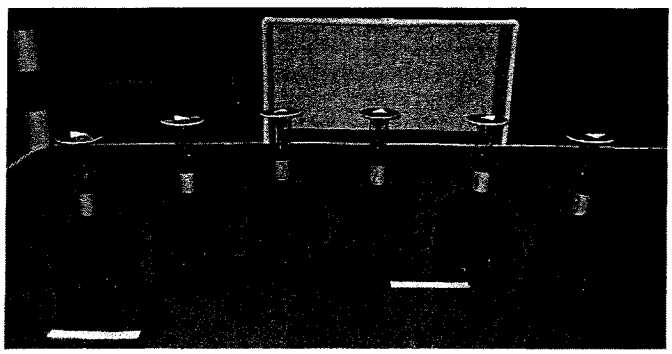


Figure 4.1 : Équipe de 6 robots joueurs de soccer développée.

4.2.1 Grandes lignes de la « Middle Size Robot League »

La « Middle Size Robot League » (MSL) constitue la ligue de prédilection dans laquelle évoluent les robots présentés ici. Parmi les différentes ligues de la *RoboCup*, elle constitue celle qui implique le plus grand nombre de problèmes à résoudre. Premièrement, les robots doivent jouer en équipe de façon autonome, ce qui est nécessaire dans toutes les ligues, mais ils doivent le faire sur un terrain aux dimensions imposantes et les règlements stipulent que tous les capteurs doivent être embarqués au sein des robots. Ainsi un système de vision global comme celui permis dans la « Small Size Robot League » ne peut être utilisé, la perception distribuée devient alors un problème très intéressant à traiter. De plus, tous les différents modules nécessaires au fonctionnement de l'équipe de robots doivent être développés (plateforme mécatronique, électronique de contrôle, capteurs, etc.), contrairement à une ligue comme la « Sony 4-Legged Robot League » qui utilise une plateforme standardisée pour toutes les équipes. Finalement, la MSL est la ligue qui se rapproche le plus du soccer joué par l'être humain, mis à part pour la plateforme, mais les développements de la « Humanoid Robot League » pourraient éventuellement s'intégrer à ceux de la MSL pour continuer les développements en vue de l'objectif de 2050.

Pour ce qui est du domaine des systèmes multi-robots, la MSL offrant la plus grande liberté au niveau des développements et le terrain le plus près d'un environnement à l'échelle humaine, elle constitue la ligue la plus intéressante dans le but de développer des systèmes multi-robots offrant des solutions génériques et applicables dans d'autres applications bien différentes.

Afin de bien saisir le contexte de la « Middle Size Robot League » il est intéressant de faire ressortir quelques données pertinentes des règlements de cette ligue. Il est à noter que ces règlements sont basés sur ceux de la Fédération Internationale de Football Association (FIFA) dans le but éventuel de les suivre intégralement. Plusieurs amendements aux règlements de la FIFA sont actuellement nécessaires pour assurer la faisabilité technique du jeu de la MSL.

Voici donc un résumé de quelques règles pertinentes :

- Une équipe peut être formée de 4 à 6 robots par équipe, dont un étant spécifiquement identifié comme le gardien de but.
- Une équipe doit fonctionner en totale autonomie durant le jeu, sans aucune intervention humaine externe, sauf celle des arbitres.
- La perception d'un robot se doit d'être en entier embarquée sur ce dernier. Aucun système de localisation utilisant des balises ajoutées dans l'environnement n'est toléré.
- Les robots d'une équipe peuvent communiquer entre eux par liaison sans fil (802.11a ou b seulement)
- Un ordinateur externe pouvant communiquer avec les robots de l'équipe est toléré.
- Les robots d'une équipe doivent démontrer qu'ils sont en mesure d'éviter les collisions avec les robots adverses et ne doivent pas volontairement causer des dommages à ces derniers.
- Les arbitres peuvent utiliser des cartons jaunes et des cartons rouges lors de différentes infractions commises par les équipes de robots.
- L'équipe de robots doit répondre, toujours de façon autonome, aux commandes de l'arbitre principal qui passent par le « Referee box », un ordinateur connecté aux deux équipes.
- Le terrain est vert et fait 8m de largeur par 12m de longueur, est délimité par des lignes blanches d'épaisseur connues et est muni de plusieurs balises colorées aux caractéristiques connues.

La Figure 4.2 présente les principales caractéristiques du terrain de la MSL. Les buts ont en fait 1m de hauteur et composé de panneaux (fond et côtés) de couleurs spécifiques.

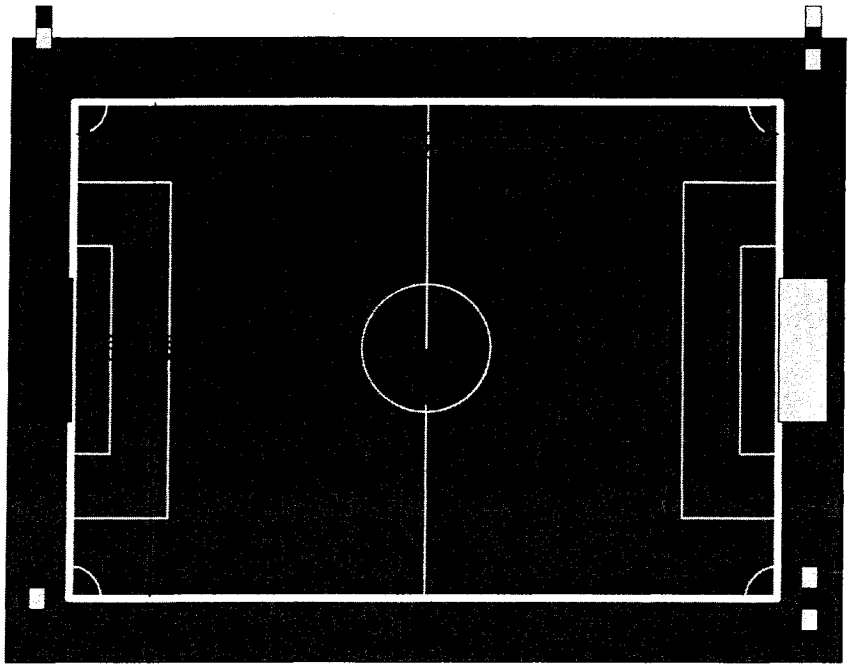
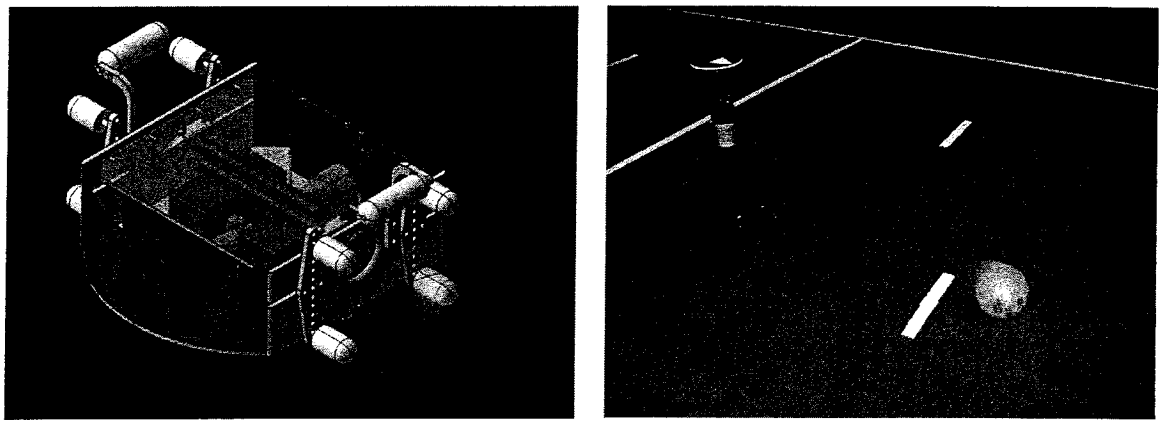


Figure 4.2 : Principales caractéristiques d'un terrain de la *RoboCup* « Middle Size Robot League ».

4.2.2 Plateforme électromécanique

La configuration des robots footballeurs est une plateforme à vitesses différentielles symétrique: 2 moteurs de propulsion et de direction couplés à deux roues motrices. Les roues motrices sont disposées au centre du robot et 2 roues libres assurant la stabilité sont disposées à l'avant et à l'arrière du robot. Le centre de masse se retrouve le plus près possible de l'axe des roues motrices. Les performances de cette plateforme sont intéressantes puisqu'elle permet d'atteindre des vitesses de près de 3m/s et 18rad/s tout en conservant des accélérations satisfaisantes. La plateforme développée est en fait une réingénierie du robot *SpinoS* développé à Polytechnique (Beaudry, [7]).Cette plateforme est complétée par un système pneumatique de contrôle et de botter du ballon. Ce système est composé de rouleaux de mousse passifs pour contrôler le ballon, en respect des règlements de la compétition, et de pistons pneumatiques permettant de botter le ballon. Ce système utilise la symétrie de la plateforme puisque qu'il comporte deux modules, à l'avant et à l'arrière du robot. La figure qui suit présente une modélisation de la plateforme sous

le logiciel CATIA utilisé pour la conception mécanique, ainsi que le botteur de ballon en action lors d'essais.



a) Conception mécanique de la plateforme b) Essai du botteur de ballon

Figure 4.3 : Développement de la plateforme électromécanique des robots.

4.2.3 Composantes électroniques et système sensoriel

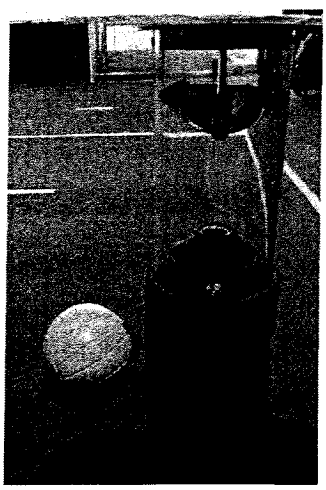
Pour obtenir un robot mobile autonome capable de percevoir, de réagir et d'agir sur son environnement et ce en temps réel, de nombreuses composantes électroniques sont nécessaires. Voici une brève description des principales composantes utilisées sur les robots.

Chacun des robots est muni d'un ordinateur embarqué de format Half-Size SBC. Cet ordinateur possède un processeur de génération Intel Pentium III, 256Mo de mémoire SDRAM, une carte graphique ainsi qu'un contrôleur Ethernet intégrés. Il se base sur les bus PC/104 et PC/104+ et plusieurs interfaces courantes (USB, RS-232, etc.) pour interfacer des cartes additionnelles ainsi que plusieurs autres composantes.

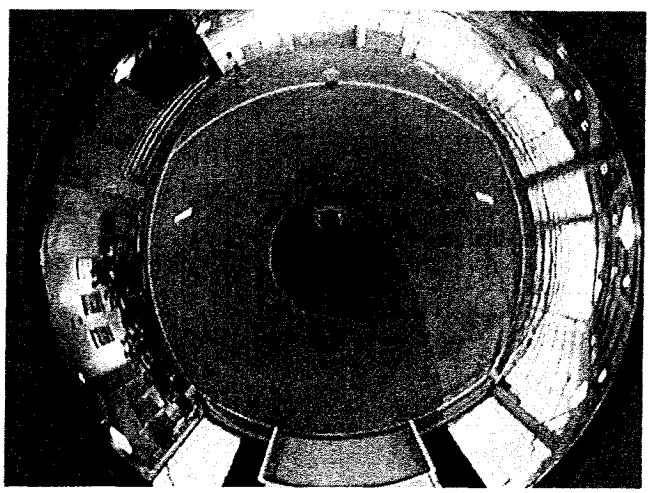
Pour commander ses moteurs ainsi que les valves pneumatiques, le robot utilise une carte de contrôle ESC629 de la compagnie RTD offrant deux boucles de rétroaction avec PID numérique ainsi que les circuits d'amplification nécessaires. De plus, cette carte offre des entrées/sorties numériques permettant entre autre de déclencher les botteurs pneumatiques au moment opportun. La carte est particulièrement bien adaptée pour un véhicule à vitesses différentielles.

Pour communiquer avec le reste de l'équipe ainsi qu'un serveur d'équipe, chacun des robots est muni d'un lien de communication sans fil utilisant la norme IEEE 802.11b. Ces liens utilisent directement la prise Ethernet de l'ordinateur embarqué et sont donc transparents au niveau du système d'exploitation.

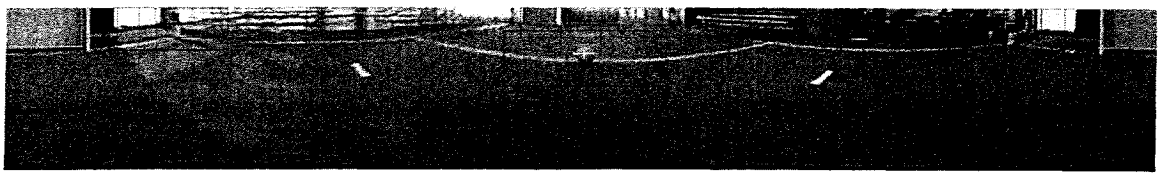
Afin de bien percevoir sa position sur le terrain ainsi que la position du ballon et des autres robots, chacun des robots est muni d'un système de perception adéquat. L'élément central de ce système de perception est le système de vision omnidirectionnel que possède chacun des robots. Il est principalement constitué d'une caméra USB et d'un miroir convexe. L'image qui suit présente l'aspect physique de ce système de vision ainsi qu'une image omnidirectionnelle vue de la caméra.



a) Miroir et caméra



b) Vue omnidirectionnelle du miroir



c) Vue panoramique obtenue pour la recherche d'objets

Figure 4.4 : Système de vision omnidirectionnel des robots joueurs de soccer.

Finalement, les robots comptent quelques autres circuits développés spécifiquement pour leurs besoins, comme un circuit d'alimentation et de recharge automatique des batteries et un circuit de commutation des valves du système pneumatique.

4.2.4 Logiciel de contrôle

Afin de réagir adéquatement, chacun des robots doit compter sur un logiciel de contrôle robuste, avec contraintes temps-réel souples et à exécution rapide et périodique. Le système d'exploitation utilisé sur les robots footballeurs est *Debian Linux* avec le noyau 2.4.18. L'ordonnanceur du système a été configuré pour fonctionner à 1kHz, ce qui donne des performances temps réel satisfaisantes. Le langage de programmation utilisé est le C++. Il est ainsi possible de développer un logiciel « multi-thread » temps-réel répondant aux besoins. Le logiciel développé modularise de façon intuitive les différentes capacités du robot (perception, cognition, action). Cette modularité a entre autre facilité le développement d'une plateforme de simulation dynamique. La figure qui suit résume la structure du logiciel de contrôle. On remarque sur ce schéma quatre classes : *Perception*, *Brain*, *MotionControl* ainsi que *Controller*.

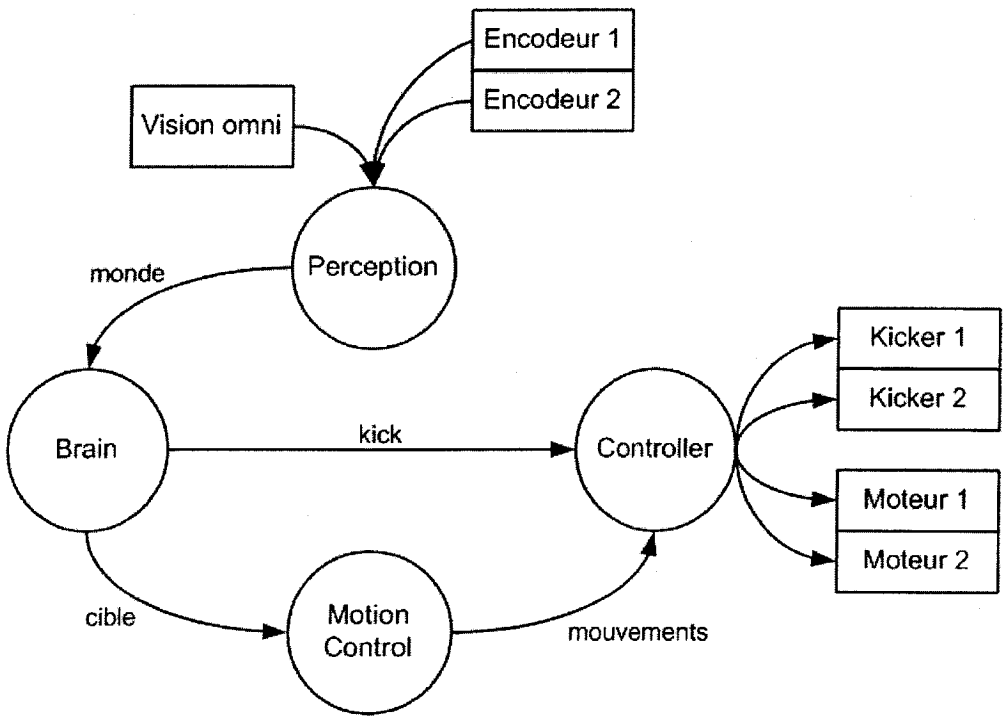


Figure 4.5 : Schéma résumant le logiciel de contrôle des robots footballeurs.

Le logiciel « multi-thread » développé utilise en fait deux boucles temporelles principales, une pour la vision au sein du module *Perception* qui fonctionne à 30 images par seconde donc 30Hz et une autre pour les autres modules (cognition et contrôle) qui fonctionne avec une période de 20ms donc 50Hz. D'autres boucles sont aussi utilisées pour l'implantation des mécanismes de communication.

4.2.4.1 Classe Brain

Elle se charge de prendre des décisions en fonction de l'état du robot et de son environnement. Par exemple, un attaquant, en fonction de son état actuel et de la représentation qu'il possède du monde (terrain, joueurs, ballon), peut prendre des décisions semblables à « tirer au but », « effectuer une passe » ou encore « jouer un rôle défensif ». Plusieurs cerveaux différents ont été programmés jusqu'à présent pour répondre à des besoins divers. Le cerveau se base sur le bon fonctionnement de la classe Perception qui lui fournit sa position et celle des autres éléments. Il utilise la classe MotionControl pour atteindre des cibles sur le terrain de jeu et il utilise la classe Controller pour botter le ballon. Cette classe est celle dans laquelle doit s'intégrer la machine décisionnelle.

4.2.4.2 Classe Perception

Ce module est responsable de prendre les informations provenant des encodeurs optiques et de la caméra pour mettre à jour toutes les informations du terrain de jeu et du robot sur ce dernier, principalement le position et vitesse du robot, du ballon et des autres robots.

4.2.4.3 Classe MotionControl

Ce module se charge principalement de déplacer le robot vers la cible désirée, à la vitesse de référence désirée, tout en évitant les obstacles du terrain. Comme on peut le voir sur le schéma, le module détermine les vitesses désirées en fonction de la cible et des éléments du terrain. La technique d'évitement d'obstacles utilise une approche réactive semblable à celle utilisée par l'équipe CMUnited de l'université Carnegie Mellon (Bowling et Veloso, [8]).

4.2.4.4 Classe Controller

Ce module a deux fonctions principales. La première est de convertir les vitesses désirées de l'espace opérationnel à l'espace articulaire et de transmettre ces vitesses désirées à la carte de contrôle. Sa deuxième fonction est d'actionner les valves pneumatiques des botteurs.

4.2.5 Intégration d'une machine décisionnelle hiérarchique

Il est possible de faire un lien entre le schéma de la Figure 4.5 et celui de la Figure 2.2. En effet, le module de perception s'y trouve sous la désignation *Perception*, le module de cognition est représenté par la classe *Brain* et finalement le module de contrôle est composé des classes *MotionControl* et *Controller*. Sachant que la machine décisionnelle hiérarchique s'intègre simplement au sein du module de cognition, c'est la classe *Brain*, et strictement cette dernière, qui va contenir la machine décisionnelle.

Le logiciel de contrôle des robots joueurs de soccer présente une certaine forme de contrôle hiérarchisé. Cela se voit rapidement en remarquant l'utilisation de deux classes distinctes pour implanter le module de contrôle, *MotionControl* étant à un niveau supérieur et utilisant la classe *Controller*. En fait, le contrôle des robots joueurs de soccer se fait en utilisant une hiérarchie composée de 3 couches comme mentionné en 2.2. La Figure 4.6 schématise l'architecture hiérarchisée du logiciel des robots. Comme le montre le schéma, la machine décisionnelle s'insère à la $3^{ème}$ couche, soit celle de la prise de décision. Le schéma montre également la division des couches 2 et 3 qui se fait au sein du module de cognition (classe *Brain*), en fait au sein de la machine décisionnelle, puisque le passage d'un nœud décisionnel à un nœud comportemental crée cette division.

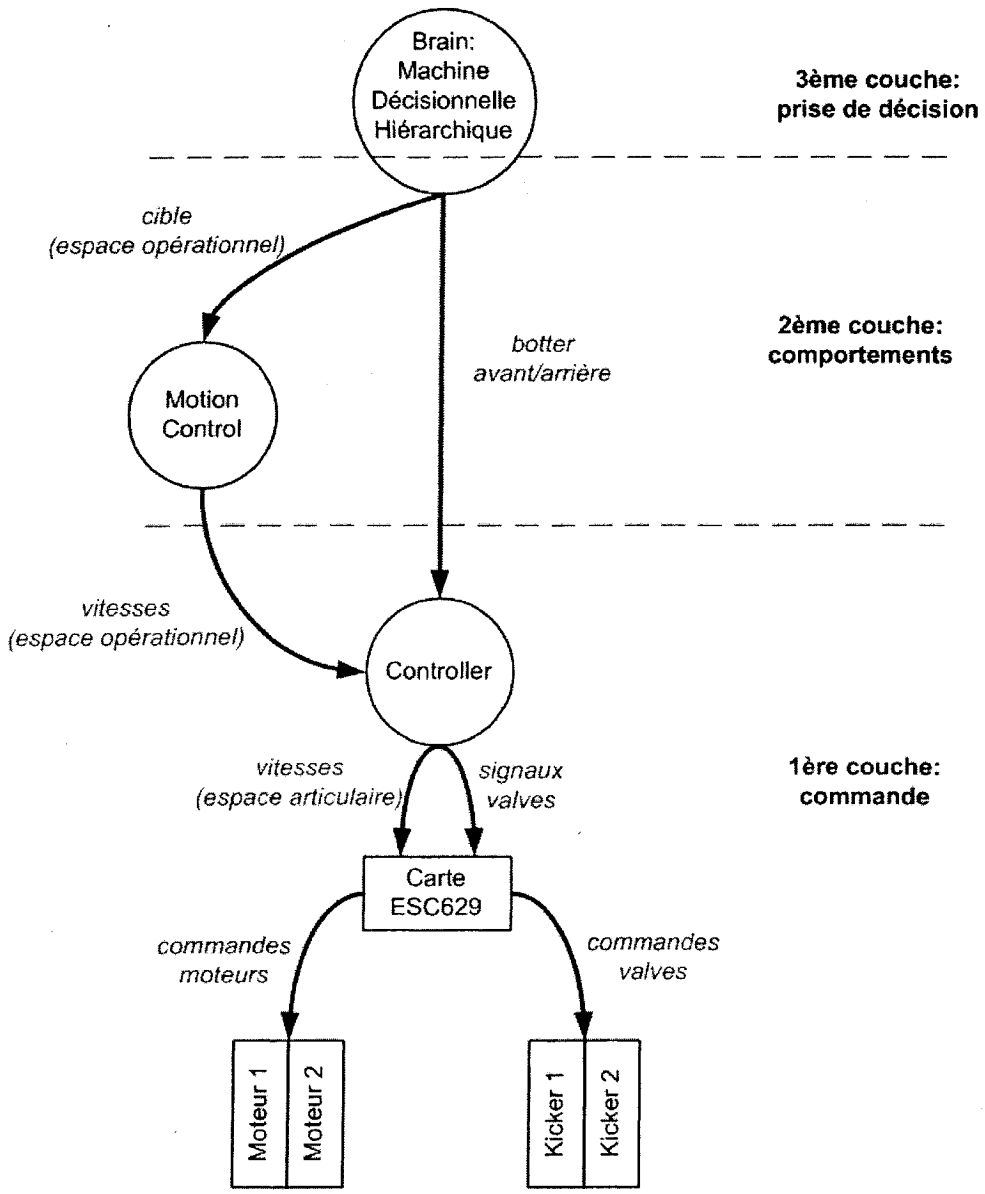


Figure 4.6 : Contrôle hiérarchisé des robots joueurs de soccer, architecture à 3 couches.

4.2.6 Structures de données pertinentes

Pour le développement des stratégies de jeu, quelques structures de données sont essentielles afin de fournir toute l'information pertinente sur la situation du jeu. Il est également important de compter sur des structures de données communes à tous les niveaux hiérarchiques, de façon à centraliser, et donc grandement simplifier, la mise à jour des informations et également faciliter le développement de la machine décisionnelle par la définition de certains standards. Il est important

donc de bien définir le contenu essentiel de ces structures de données et d'en permettre la mise à jour éventuelle lorsque de nouvelles informations jugées utiles doivent y être ajoutées.

Pour les robots joueurs de soccer, deux structures de données contenant l'essentiel des informations nécessaires au développement de la machine décisionnelle sont utilisées : *Robot_info* et *Field_info*. Ces structures sont résumées dans le Tableau 4.1. *Robot_info* contient donc les données propres au robot et *Field_info* contient les données propres au terrain de jeu et à son contenu.

Structures	***Paramètres***
Données propres au robot *(Robot_info)*	• Cible position-orientation sur le terrain en m et rad (*Field_pos target*) • Position et orientation du robot sur le terrain (*rbt_pos*) • Vitesses du robot sur le terrain (*vtan*, *omega*) • Identité numérique du robot (*ID*) • Équipe du robot (*team*)
Données du terrain de jeu (*Field_info*)	• Position et orientation du ballon sur le terrain (*ball_pos*) • Vitesses du ballon sur le terrain (*ball_v*, *ball_dir*) • Positions et orientations des coéquipiers sur le terrain (*teammates[(NB_PLAYERS/2)-1]*) • Positions et orientations des adversaires sur le terrain (*opponents[NB_PLAYERS/2]*) • Dimensions du terrain (*field_w*, *field_l*) • Dimensions des zones de but et des buts (*zone_w*, *zone_l*, *goal_w*) • Fonction de mesure de distance (*distance(pos1, pos2)*)

Tableau 4.1 : Structures de données essentielles du logiciel de contrôle des robots joueurs de soccer.

4.2.7 Architecture de communication

Jusqu'à présent le logiciel présenté concerne un seul robot pris individuellement. Mais il est rare qu'un système multi-robots puisse fonctionner sans aucune communication et l'équipe de robots joueurs de soccer développée ne fait pas exception. Ainsi, de façon à assurer une coopération entre les robots qui est satisfaisante mais surtout pour respecter les règles du jeu de la « Middle Size Robot League », une communication au sein de l'équipe est nécessaire.

La communication des robots joueurs de soccer utilise une approche client/serveur qui permet d'interconnecter de façon modulaire plusieurs différents modules ayant des fonctions spécifiques. La Figure 4.7 présente l'architecture de communication développée pour l'équipe de robots joueurs de soccer. Cette architecture correspond au modèle a) de la Figure 3.8.

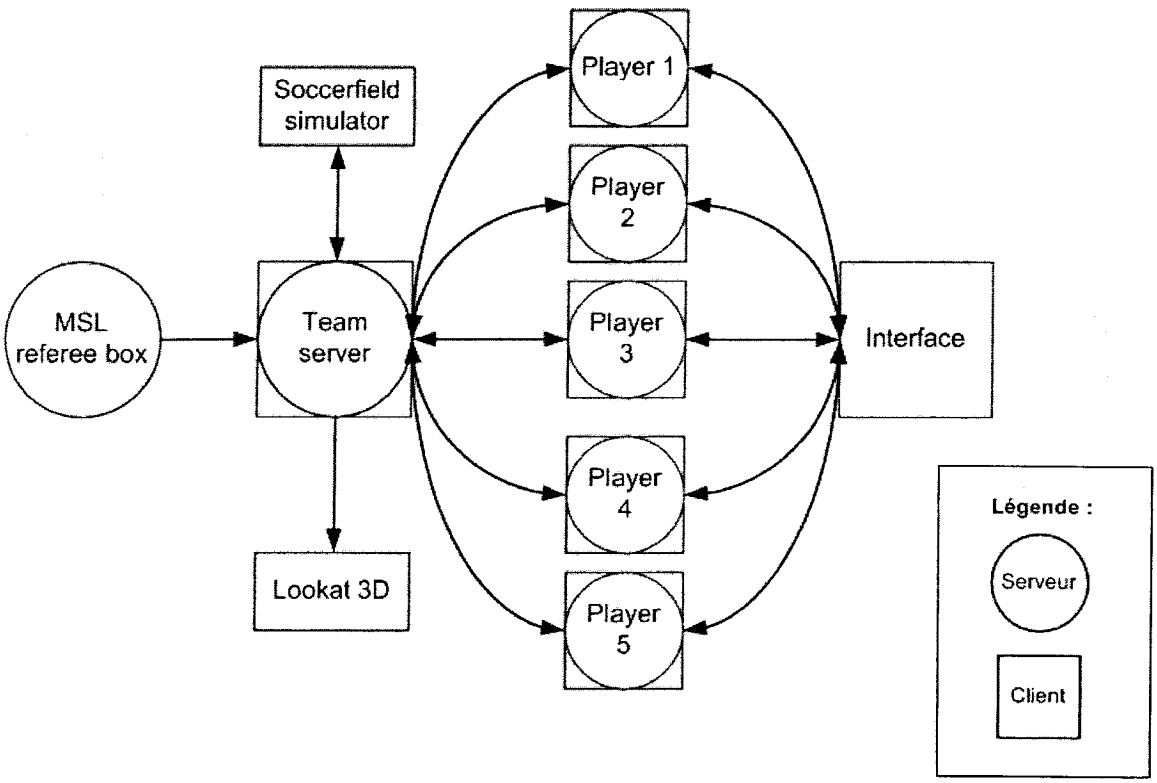


Figure 4.7 : Schéma de l'architecture Clients/Serveurs du système multi-robots.

Le schéma présente donc les divers modules interconnectés. En premier lieu, on remarque que les robots (*Player 1* à *Player 5*) de l'équipe sont à la fois des clients et des serveurs. Le nombre de robots interconnectés par cette architecture peut varier, mais en compétition 5 robots seulement sont utilisés. Une interface de contrôle du système (*Interface*) se connecte, en mode client, à chacun des serveurs présents sur les robots. Cette interface est très utile pour connaître l'état du système et le contrôler mais en situation normale de jeu, en mode autonome, l'utilisation de cette interface pourrait être à la limite interdite par les règlements de la MSL. La communication pour répondre aux besoins du jeu comme tel se fait grâce à un serveur d'équipe central (*Team server*) dont chaque robot en est un client. Ce serveur est alors en mesure de partager des informations

entre les robots mais surtout d'implanter des mécanismes de supervision décisionnelle. Au serveur d'équipe peuvent également se connecter, toujours en mode client, un visualisateur virtuel (*Lookat 3D*) ainsi qu'un simulateur de terrain de jeu (*Soccerfield simulator*), qui simule en fait l'interaction entre les différents robots et le ballon sur le terrain. Finalement, afin de recevoir les directives de l'arbitre qui contrôle un match officiel de la MSL, le serveur d'équipe se doit, et cette fois en mode client, d'être connecté à l'ordinateur servant de « referee box » (*MSL referee box*).

Les clients/serveurs utilisés dans cette architecture sont implantés en utilisant des « sockets » TCP/IP. En fait, la librairie *MICROB* (Institut de recherche d'Hydro-Québec, [21]) est utilisée car elle offre toutes les fonctionnalités permettant d'utiliser de tels clients/serveurs. Cependant, une autre version de clients/serveurs utilisant aussi les « sockets » TCP/IP a été développée par Sylvain Marleau de l'équipe *Robofoot ÉPM*. Cette version est utilisée pour les communications entre l'*Interface* et les robot. Éventuellement une seule version standardisée devrait être utilisée et il est préférable pour ce genre de fonctionnalités de compter le plus possible sur des librairies génériques utilisées par un grand nombre d'utilisateurs.

4.2.8 Plateforme de simulation

La modularité du logiciel de contrôle ainsi que les possibilités de la librairie *MICROB* utilisée dans le projet ont grandement facilité le développement d'une plateforme de simulation dynamique. En fait, au niveau du logiciel de contrôle, seule la classe *Controller* doit être modifiée où les accès à la carte de contrôle sont remplacés par les modèles cinématique et dynamique de la plate-forme à vitesse différentielles (DeSantis, [14]). Un simulateur de la dynamique du ballon ainsi qu'un visualisateur 3D, présenté à la figure qui suit, permettent de compléter ce simulateur.

Les apports d'une telle plateforme de simulation sont importants lorsque vient le temps de développer des algorithmes de contrôle de haut niveau ainsi que les différents comportements des robots, car il est alors possible de le faire sans nécessiter le système réel. La machine décisionnelle des robots en vue des compétitions a presqu'entièrement été développée en simulation. Seuls les comportements demandent beaucoup d'essais sur le terrain réel puisque le modèle dynamique de simulation des joueurs néglige certains aspects comme le module de contrôle de ballon. En fin de compte, la couche des comportements demande l'utilisation assez

fréquente du système réel tandis que la couche de prise de décision peut se développer entièrement en simulation. Idéalement, le modèle dynamique des robots serait raffiné de façon à permettre également l'ajustement précis des comportements par voie de simulation.

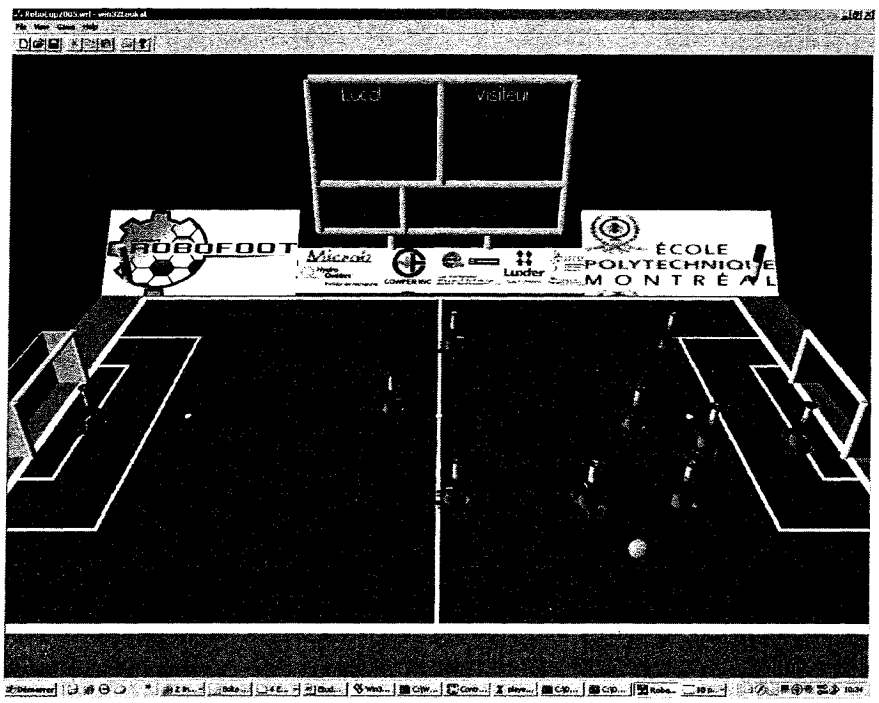


Figure 4.8 : Visualisateur virtuel utile en simulation.

4.3 Machine décisionnelle pour équipe de robots joueurs de soccer

La machine décisionnelle développée pour l'équipe de robots joueurs de soccer doit permettre à un robot, pris individuellement, de jouer au soccer de façon autonome. De plus, cette machine décisionnelle doit permettre aux robots d'une même équipe de coordonner leur prise de décision de façon à établir un jeu d'équipe coopératif. Sachant que la machine décisionnelle est implantée de façon distribuée, c'est-à-dire que chaque robot de l'équipe possède sa propre machine décisionnelle fonctionnant en temps réel, il est possible de prévoir des mécanismes coopératifs au sein même de la prise de décision et sans supervision ni communication explicite. Ces mécanismes assureront un fonctionnement adéquat de l'équipe dans le cas où la communication entre les robots et le serveur n'est pas possible. Cependant, comme la communication est

disponible pour la grande majorité du temps, des mécanismes de supervision peuvent être développés de façon à améliorer les capacités de coopération de l'équipe.

L'équipe de robots que la machine décisionnelle doit gérer est composée de 5 robots (dont un gardien de but), mais ce nombre peut cependant varier de façon impromptue pour passer à 4 ou 6 robots. La machine décisionnelle sera donc développée pour un fonctionnement performant à 5 robots, mais il est aussi possible d'ajouter ou de retirer un robot en quelques changements mineurs sans affecter le fonctionnement général de l'équipe.

L'équipe de robots joueurs de soccer étant développée pour évoluer dans la catégorie « Middle Size Robot League » de la *RoboCup*, il est important qu'elle respecte tous les règlements appliqués lors des matchs en tournoi (MSL Technical Committee, [26]). Parmi ceux-ci, plusieurs règles influencent le développement d'une machine décisionnelle, par exemple l'obligation d'avoir un système multi-robots entièrement autonome, les seules interactions avec les développeurs étant possibles lorsqu'un robot se trouve hors du terrain. Et comme au soccer joué par les humains, il existe des règles précises d'arrêts de jeu (remise en touche, coup de pied de coin, etc.) et l'équipe doit donc gérer plusieurs « cas spéciaux ».

4.3.1 Structure générale de la machine décisionnelle

De façon générale, la hiérarchie décisionnelle des robots joueurs de soccer est munie au maximum de 6 étages, 4 étages de nœuds décisionnels, un étage de nœuds comportementaux et un étage d'actions. Dans un premier temps la décision principale se charge de choisir du bon mode de jeu de façon à permettre à l'équipe de jouer au soccer comme tel et aussi de répondre aux nombreux cas spéciaux. Le nombre d'étages qui viennent à la suite du mode sont variables en fonction du mode courant. Il est effectivement possible qu'il y ait, pour certains modes et choix subséquents, moins de 6 étages que ceux nécessaires au jeu de soccer. Si nous considérons un mode où le nombre d'étages est complet et égal à 6, la décision qui suit celle du mode est celle du patron de jeu à utiliser en fonction de la situation actuelle sur le terrain. Il y a ainsi un certain nombre de patrons de jeu qui peuvent être utilisés. Chacun de ces patrons doit ensuite être composé d'un nombre de nœuds décisionnels égal au nombre de robots dans l'équipe, excluant et qui correspondent en fait aux différents rôles que joueront chacun des robots de l'équipe. Ensuite, pour un rôle donné, il y aura un certain nombre de comportements qui permettront à chaque robot

de bien jouer son rôle au sein de l'équipe. Finalement, comme le définit le concept de machine décisionnelle hiérarchique, chacun des comportements mène à un ensemble d'actions. Ces étages sont schématisés à la Figure 4.9 et les paragraphes qui suivent décrivent en détail la machine décisionnelle développée pour l'équipe de robots footballeurs.

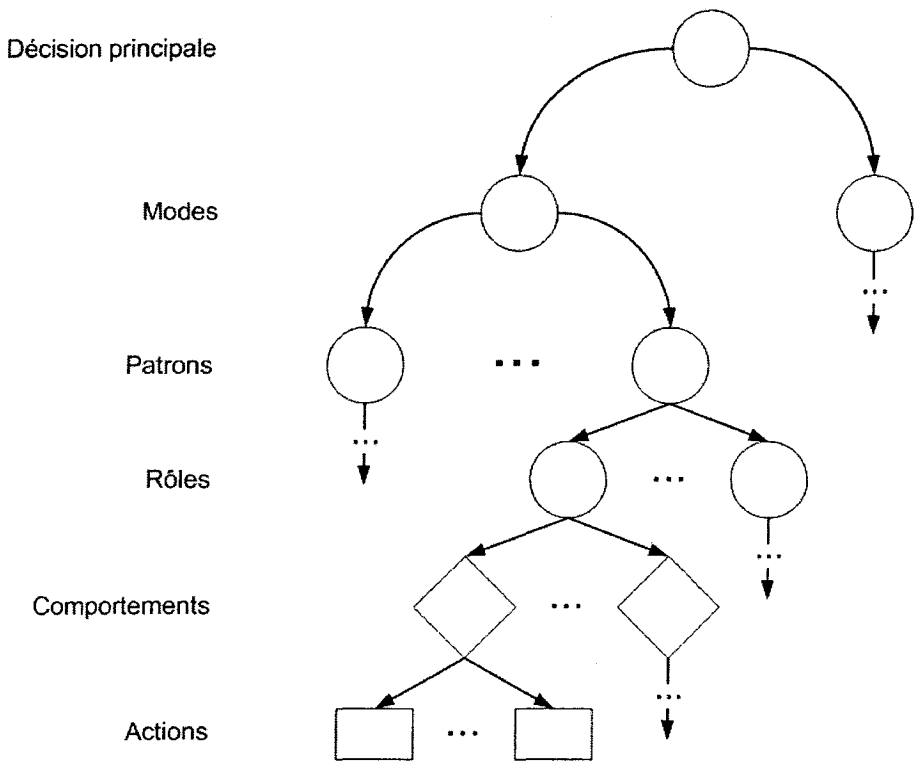


Figure 4.9 : Structure générale de la hiérarchie décisionnelle des robots footballeurs

4.3.2 Machine décisionnelle : *Soccerteam_HDM*

La Figure 4.10 présente la machine décisionnelle développée pour l'équipe de robots joueurs de soccer. Afin d'alléger la représentation de la machine dans son ensemble, tous les comportements utilisés de façon similaires par un groupe de rôles sont regroupés, et des acronymes ont été utilisés pour représenter les nœuds et les actions. Sinon il aurait été impossible de présenter l'ensemble de la machine en un seul schéma. Il apparaît évident qu'il est difficile de s'y retrouver clairement et donc pour la présentation détaillée de la machine le schéma de l'ensemble sera fractionné.

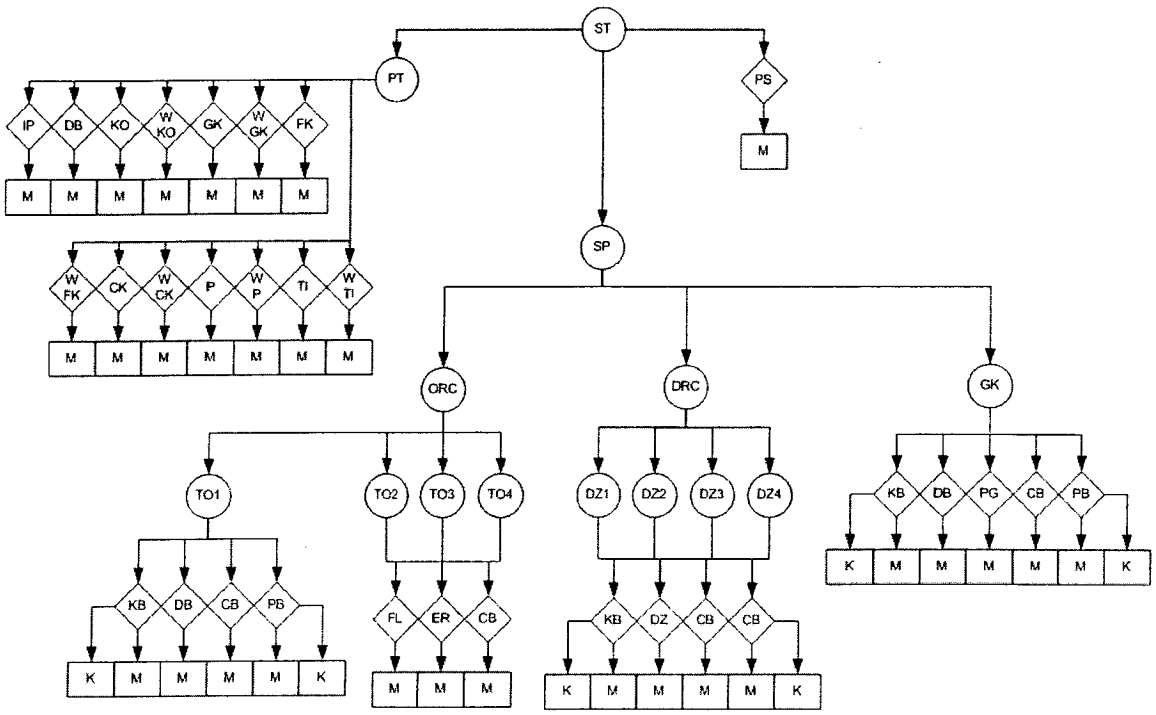


Figure 4.10 : Machine décisionnelle hiérarchique des robots joueurs de soccer.

En observant le deuxième étage de la hiérarchie, l'étage du mode, on constate qu'il est composé de trois nœuds distincts. La machine *Soccerteam_HDM* comporte donc trois modes qui sont présentés à la Figure 4.11 qui suit. Le mode *PrepareTeam* permet de positionner les joueurs de l'équipe en fonction aux différents arrêts de jeu. Le mode *SoccerPlayer* est le mode le plus complexe qui implémente tous les éléments nécessaires pour l'obtention d'une équipe de robots joueurs de soccer autonome. Finalement, le mode *PlayerStop* est un mode d'arrêt qui permet de stopper tous les joueurs de l'équipe lorsque nécessaire. On remarque, que comparativement aux deux premiers modes, le mode *PlayerStop* est beaucoup plus simple puisqu'il n'est composé que d'un seul nœud comportemental menant à une action. Aucune décision n'est donc prise au sein de ce mode.

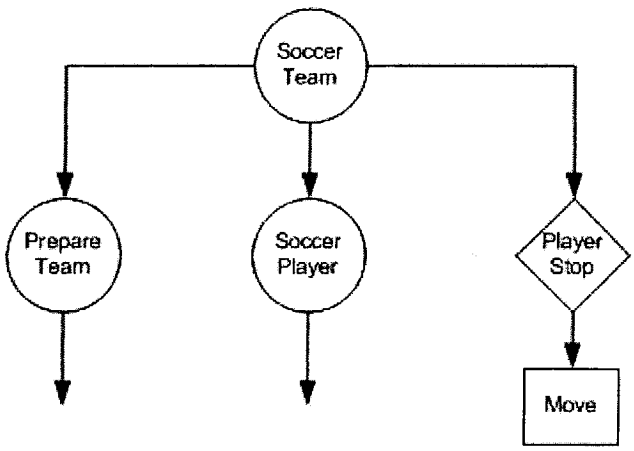


Figure 4.11 : Choix du mode pour la machine décisionnelle *SoccerTeam_HDM*.

À la Figure 4.11 il y a un mode qui présente directement un nœud comportemental et ainsi une action correspondante. Avec la configuration électromécanique des robots joueurs de soccer, il a été possible de définir simplement deux actions distinctes permettant d'utiliser à bon essor tous les actuateurs des joueurs. Ainsi, la machine décisionnelle des joueurs de soccer peut compter sur les actions *Move* et *Kick*. La plupart des comportements développés n'ont accès seulement à *Move*, mais certains ont accès à *Move* et à *Kick*.

Actions	***Paramètres***
Move	• Cible position-orientation sur le terrain en m et rad (*Field_pos target*) • Vitesse de croisière en m/s (*double vtan*) • Vitesse finale sur la cible en m/s (*double vtanf*) • Activation/désactivation de l'évitement d'obstacles (*int correct_target*) • Activation/désactivation de l'évitement du ballon comme obstacle (*int ball_avoid*)
Kick	• Type de coup de pied (*int kick*): *NO_KICK*, *FRONT_KICK*, *BACK_KICK* ou *BOTH_KICK* • Durée du coup de pied en ms (*int kick_time*)

Tableau 4.2 : Actions définies pour la machine décisionnelle *Soccerteam_HDM*.

4.3.3 Mode préparation pour arrêts de jeu : *PrepareTeam*

Le mode *PrepareTeam* permet de préparer l'équipe lors des arrêts de jeu, c'est-à-dire que chacun des joueurs prend une position déterminée dynamiquement qui dépend de l'arrêt de jeu courant et également de la position du ballon et des adversaires. Ce mode permet donc de positionner les joueurs adéquatement sur le terrain en vue de la reprise du jeu. La Figure 4.12 présente un exemple de positionnement possible, le cas d'un « corner » attribué à l'équipe en zone adverse.

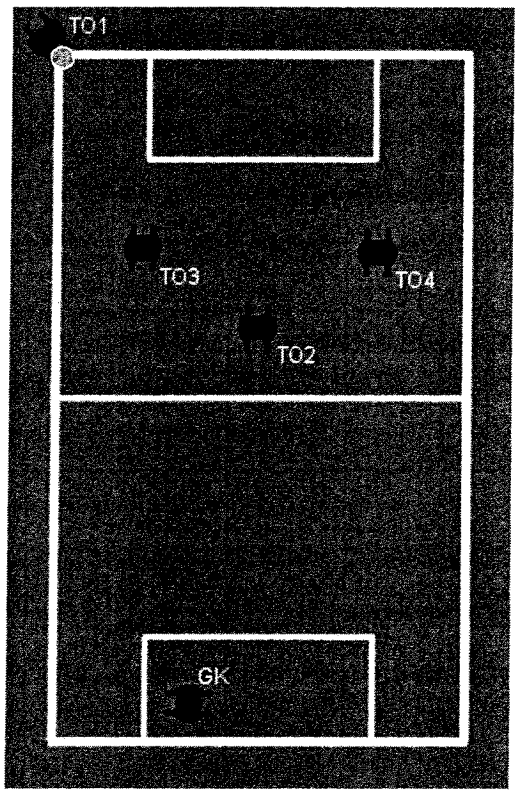


Figure 4.12 : Exemple de positionnement dans le cas d'un "corner" attribué à l'équipe de joueurs.

La Figure 4.13 présente l'ensemble des patrons associés au mode *PrepareTeam*. Il y a en fait 13 patrons répondant aux 13 possibilités d'arrêts de jeu définies dans les règlements de la MSL, et également un quatorzième patron, *InitPos*, qui est utilisé lors de l'initialisation d'une période de jeu. Chacun des patrons utilise en réalité le même nœud comportemental, le nœud *PlacePattern*. La seule action possible pour ce comportement est *Move* puisqu'aucune situation de demandera de botter le ballon.

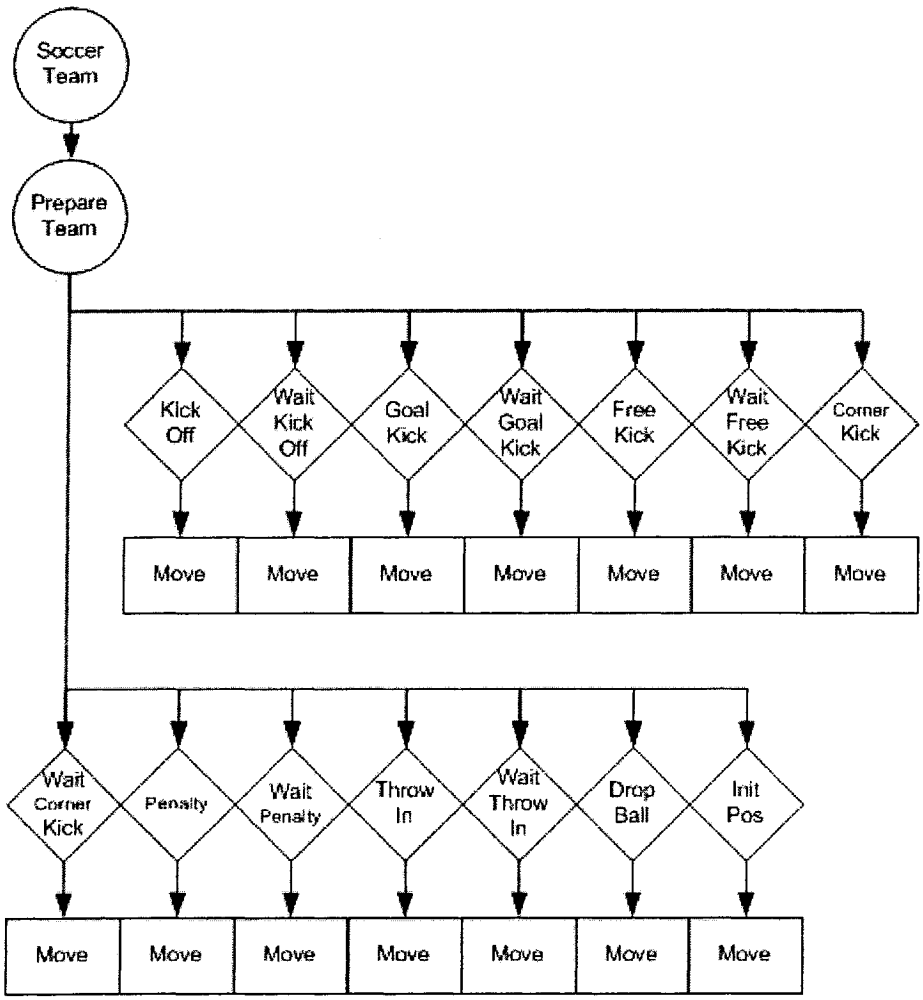


Figure 4.13 : Définition du mode *PrepareTeam*.

Il est important de mentionner qu'étant donnée l'architecture de communication utilisée pour le système de robots joueurs de soccer, il est impossible que chaque robot puisse, individuellement, prendre une quelconque décision au sein de ce patron. En effet, puisque le « referee box » est connecté au serveur d'équipe (voir Figure 4.7), et non directement aux robots, c'est ce dernier qui reçoit les indications d'arrêts de jeu. C'est donc ce dernier qui impose par voie de supervision décisionnelle le mode *PrepareTeam* ainsi que le patron à adopter (voir 4.4).

4.3.4 Mode joueur : *SoccerPlayer*

Le mode *SoccerPlayer* implémente toutes les fonctionnalités des robots joueurs de soccer lorsqu'ils sont en situation de jeu comme tel. Comme le présente la Figure 4.9, au-dessous de la hiérarchie du mode se trouve les patrons de jeu. Au sein du mode *SoccerPlayer* se trouvent 3

différents patrons de jeu : un patron de jeu offensif (*OffenseRC*), un patron de jeu défensif (*DefenseRC*) ainsi qu'un patron spécifique au gardien de but (*GoalKeeper*).

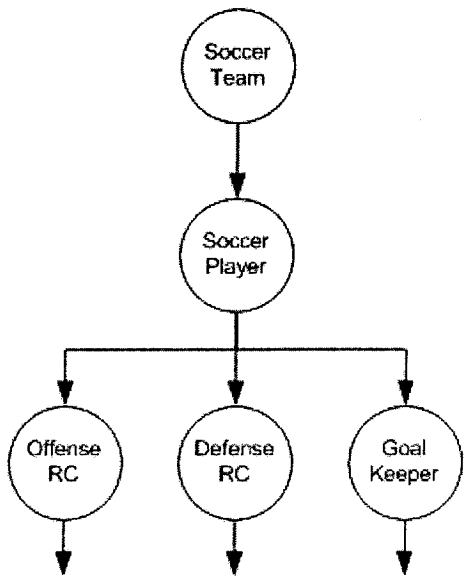


Figure 4.14 : Patrons de jeu associés au mode *SoccerPlayer*.

Au départ, plusieurs patrons différents ont été imaginé et certains développés et l'objectif était de sélectionner parmi plusieurs patrons disponibles le meilleur à appliquer dans une certaine situation. Il était aussi envisagé de mesurer en temps réel des mesures de performance des patrons (par exemple un but marqué pour un patron offensif, ou encore le temps de possession du ballon) permettant de faire évoluer au cours d'un match des poids de sélection des patrons de façon à s'adapter au jeu d'une certaine équipe et qu'il y ait donc une forme d'apprentissage au niveau des robots ou du superviseur décisionnel pour ce qui est du choix de patron. Cependant, le temps demandé par le développement de divers modules essentiels en vue des compétitions et par conséquent le manque de temps pour tester les différents patrons de jeu en situation réelle, de même que le désir d'obtenir un système au comportement robuste et prévisible, ont justifié l'utilisation d'un seul patron offensif et d'un seul patron défensif. Néanmoins, des travaux reliés à l'étude de divers mécanismes d'apprentissage devraient être entrepris, comme par exemple l'utilisation de « Reinforcement Learning », concept fréquemment utilisé dans le domaine du soccer robotisé (Takahashi et coll., [40]).

Toutes les décisions au sein du mode *SoccerPlayer* sont prises de façon distribuée, c'est-à-dire en temps réel sur chacun des robots. Ces décisions sont prises selon des règles préétablies, ce qui ressemble beaucoup au concept de « Locker-room agreement » de Peter Stone (Stone, [39]). Le choix du patron *GoalKeeper* se fait strictement selon l'identité du robot. Comme le stipule le règlement de la MSL, un robot est identifié comme étant le gardien et donc ce robot choisira toujours le patron de gardien et aucun autre ne le fera. Pour le choix des patrons offensif ou défensif, le choix se fait selon une règle commune très simple :

⇨ **SI** aucun robot de l'équipe ne se trouve à moins de 1,5m du ballon
OU
le ballon se trouve à 1m et moins en zone défensive
⇨ **ALORS** : sélection du patron défensif

⇨ **SINON SI** au moins un robot se trouve à moins d'1m du ballon
ET
le ballon se trouve à 1m et plus en zone défensive
⇨ **ALORS** : sélection du patron offensif

⇨ **SINON** : sélection du patron précédent

Cette règle a été ajustée par expérimentation et ses paramètres sont ajustables en fonction par exemple de la dimension du terrain. La règle peut sembler très défensive et tel est le but de façon à éviter les situations où l'équipe se compromet. Néanmoins, même en patron défensif, la possibilité d'utiliser certains comportements offensifs est disponible et donc il ne s'agit pas ici d'empêcher toute initiative offensive.

Puisque les conditions de sélection de patron ne dépendent que des positions sur le terrain. La prise de décision distribuée devrait normalement mener à ce que tous les robots soient toujours sur le même patron de jeu. Cependant, étant donnée la perception distribuée qui ne peut être autrement qu'imparfaite, il est inévitable que certains conflits décisionnels surviennent.

4.3.4.1 Patron offensif : *OffenseRC*

Le patron *OffenseRC* permet donc d'implémenter la partie offensive du jeu des robots. Comme il y a 4 robots disponibles pour établir les patrons de jeu (1 des 5 robots étant désigné comme gardien), ce patron est constitué de 4 rôles distincts comme le montre la Figure 4.15 : *TOffense1*, *TOffense2*, *TOffense3* et *TOffense4*.

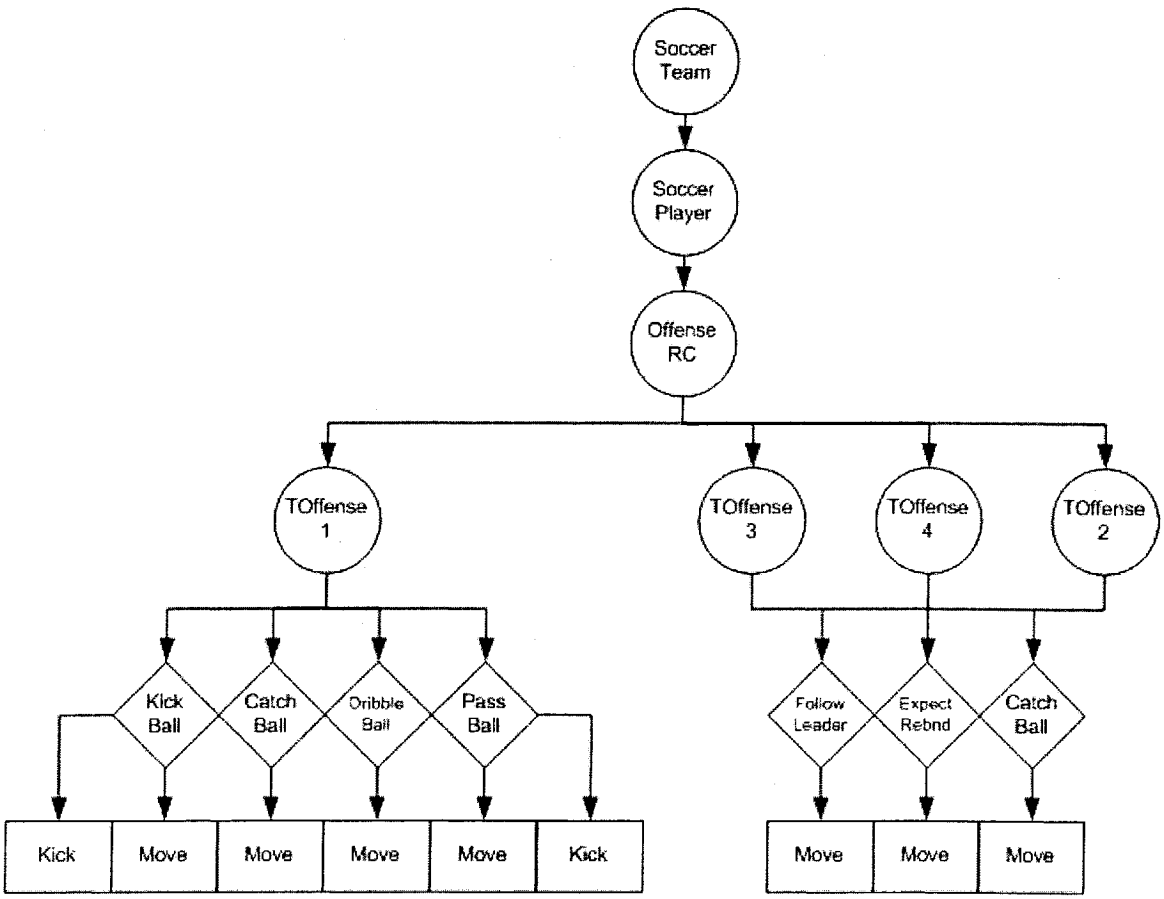


Figure 4.15 : Définition du patron de jeu *OffenseRC*.

Ce patron de jeu définit premièrement un rôle de leader (*TOffense1*) qui vise principalement à prendre le contrôle du ballon et à marquer un but. Ensuite, trois rôles de suiveurs (*TOffense2*, *TOffense3*, *TOffense4*) sont définis dans le but de supporter l'attaque tout en ne compromettant pas la tâche défensive en cas de revirement. La Figure 4.16 présente la formation définie par ce patron de jeu.

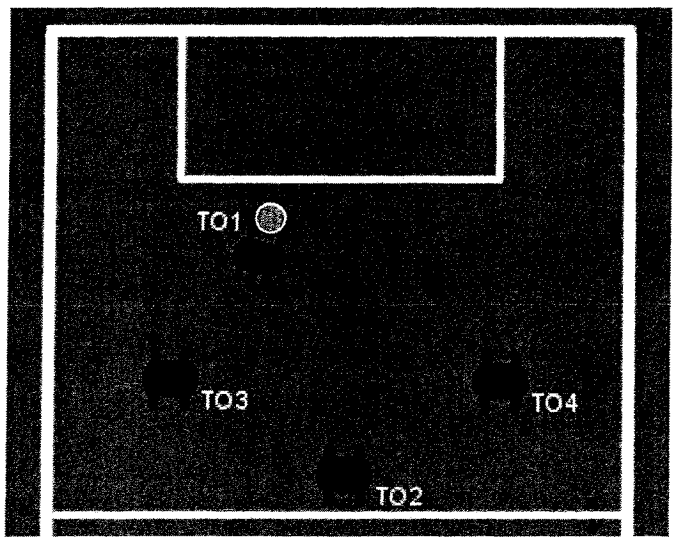


Figure 4.16 : Formation du patron offensif *OffenseRC*.

Ces rôles utilisent en fait seulement deux rôles de base différents, *TOffenseL* pour le rôle du leader et *TOffenseF* pour les rôles de suiveurs. Voici un résumé de ces rôles :

- *TOffenseL* : joueur qui a comme objectif de prendre possession du ballon (comportement *CatchBall*), dribbler avec celui en progressant vers le but adverse (comportement *DribbleBall*) et de tenter un tir au but lorsque la situation est favorable (comportement *KickBall*). Il est également en mesure de passer la balle (comportement *PassBall*) lorsqu'il juge qu'un coéquipier est en meilleure position que lui (peu probable compte tenu du patron défini mais tout de même possible).
- *TOffenseF* : joueur qui a comme objectif de suivre le leader (comportement *FollowLeader*). Ce joueur peut également se positionner pour obtenir un retour de lancer (comportement *ExpectRebound*) ou encore chercher à prendre possession du ballon s'il est libre près de lui (comportement *CatchBall*).

Pour le rôle *TOffenseF*, le décalage ainsi que la position du leader peuvent être définis par la hiérarchie supérieure. Dans le cas du patron *OffenseRC*, le leader utilisé est en fait la position du ballon pour deux raisons : cette position est plus stable et plus précise au niveau de la perception, ce qui cause moins de mouvements saccadés de la part des suiveurs, et la fait d'utiliser la ballon assurer un retrait défensif beaucoup plus efficace, car le ballon peut parfois se retrouver derrière le leader. Pour ce qui est des décalages, le patron *OffenseRC* ne fait qu'en spécifier trois différents

pour les trois suiveurs. D'ailleurs, ces décalages sont variés en fonction de la position du ballon sur le terrain, le triangle observé à la Figure 4.16 peut donc varier.

Brièvement, le leader est établi en fonction de sa position relative au ballon favorable par rapport à ses coéquipiers. Pour les rôles des suiveurs, ils sont établis en fonction de la position de chacun des coéquipiers sur le terrain (le plus près d'une zone de couverture prend cette zone). Chacun des coéquipiers évalue de façon distribuée ces conditions et ainsi les rôles sont alloués dynamiquement. Un suiveur sera donc en mesure de devenir leader dans le cas où ce dernier perd possession du ballon. Tout comme pour le choix du patron, cette prise de décision tient compte des positions sur le terrain et peut mener à des conflits décisionnels.

4.3.4.2 Patron défensif : *DefenseRC*

Par opposition au patron offensif, le patron D*efenseRC* permet donc d'implémenter la partie défensive du jeu des robots. Comme il y a 4 robots disponibles pour établir les patrons de jeu (1 des 5 robots étant désigné comme gardien), ce patron est constitué de 4 rôles distincts comme le montre la Figure 4.17 : *DefenseZone1*, *DefenseZone2*, *DefenseZone3* et *DefenseZone4*.

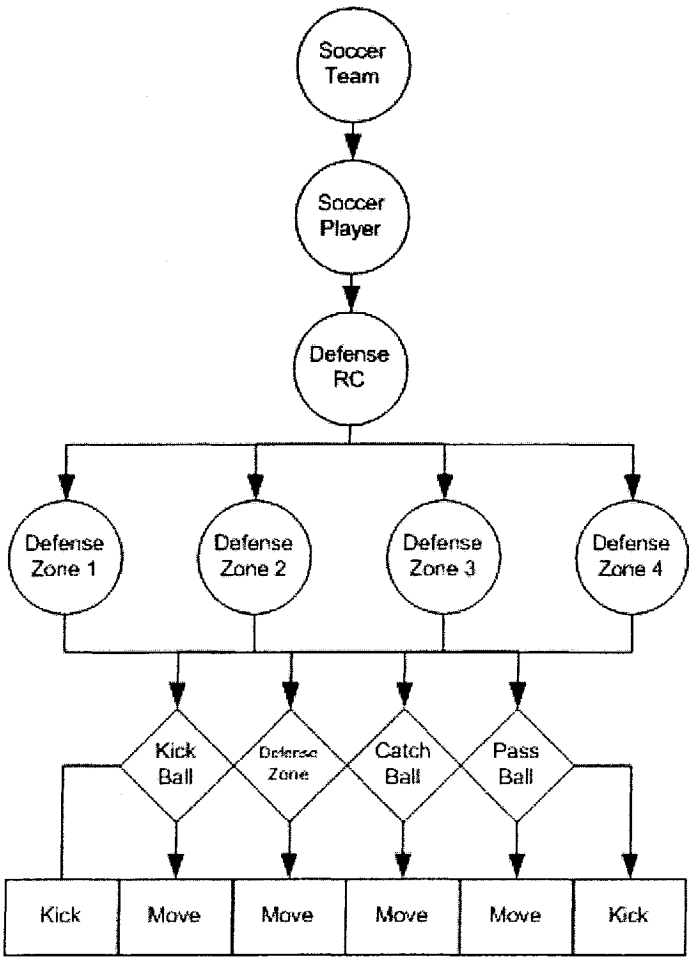


Figure 4.17 : Définition du patron de jeu *DefenseRC*.

Ce patron de jeu définit 4 rôles qui permettent de couvrir la zone défensive en entier de façon à bloquer le porteur du ballon (joueur adverse), à couvrir les autres joueurs adverses qui pénètrent en zone défensive et finalement à permettre la reprise de possession du ballon pour retourner en patron offensif. La Figure 4.18 présente la formation définie par ce patron de jeu.

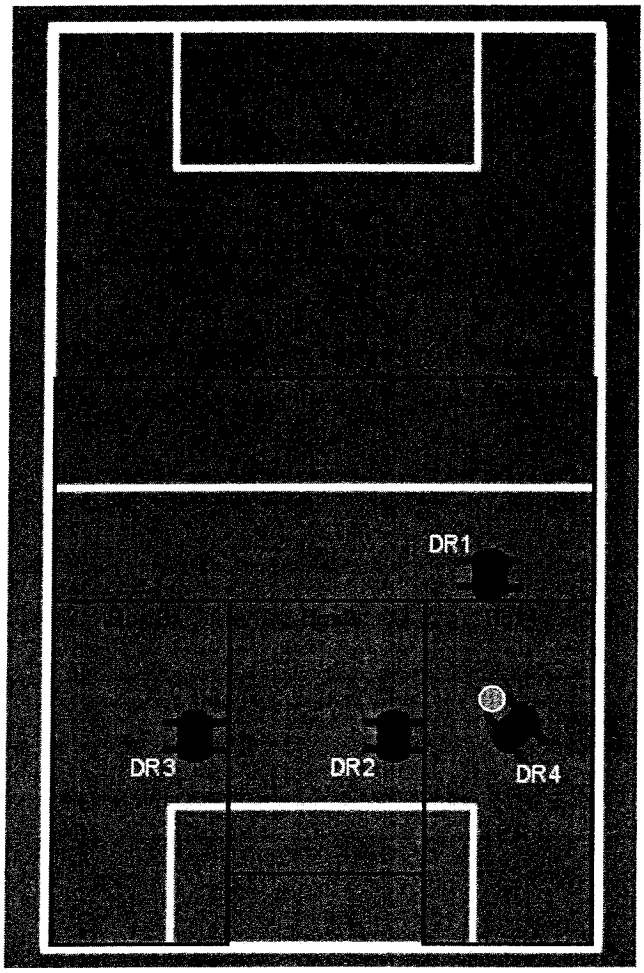


Figure 4.18 : Patron défensif *DefenseRC*.

Chacun des 4 rôles défensifs définis se base sur le même rôle : *DefenseZoneR*. En voici un résumé.

- *DefenseZoneR* : joueur jouant un rôle défensif et couvrant une zone rectangulaire. Il se positionne dans sa zone de façon à en rendre difficile la pénétration par un joueur adverse de la zone défensive par les joueurs adverses et le ballon (comportement *Protect*). Lorsqu'il est près suffisamment près du ballon et qu'il le juge opportun, il tente de prendre possession du ballon (comportement *CatchBall*). S'il prend possession du ballon il peut tenter un tir au but adverse (comportement *KickBall*) ou encore une passe à un coéquipier (comportement *PassBall*).

Les comportements plus offensifs *KickBall* et *PassBall* peuvent être bien utilise pour ce rôle lorsqu'un robot prend possession du ballon mais que le patron demeure tout de même défensif. Selon les conditions du mode *SoccerPlayer* cela ne devrait cependant pas être le cas. Pour ce qui est de la définition des zones de couvertures rectangulaires, elles peuvent être définies par la hiérarchie supérieure et le patron *DefenseRC* se charge donc de définir les zones présentées à la Figure 4.18. Les rôles de ce patron sont également alloués dynamiquement et de façon distribuée en fonction des positions respectives des coéquipiers sur le terrain.

4.3.4.3 Patron du gardien : *GoalKeeper*

En vue de la RoboCup 2005, le gardien de but a été physiquement adapté pour bien remplir sa tâche, présentant ainsi un aspect extérieur différent des autres robots de l'équipe. La Figure 4.19 présente l'aspect physique de ce gardien. On peut remarquer une base élargie ainsi que des panneaux translucides de polycarbonate, dans le but de bloquer efficacement les tirs au sol ainsi que les tirs aériens effectués par certaines équipes.

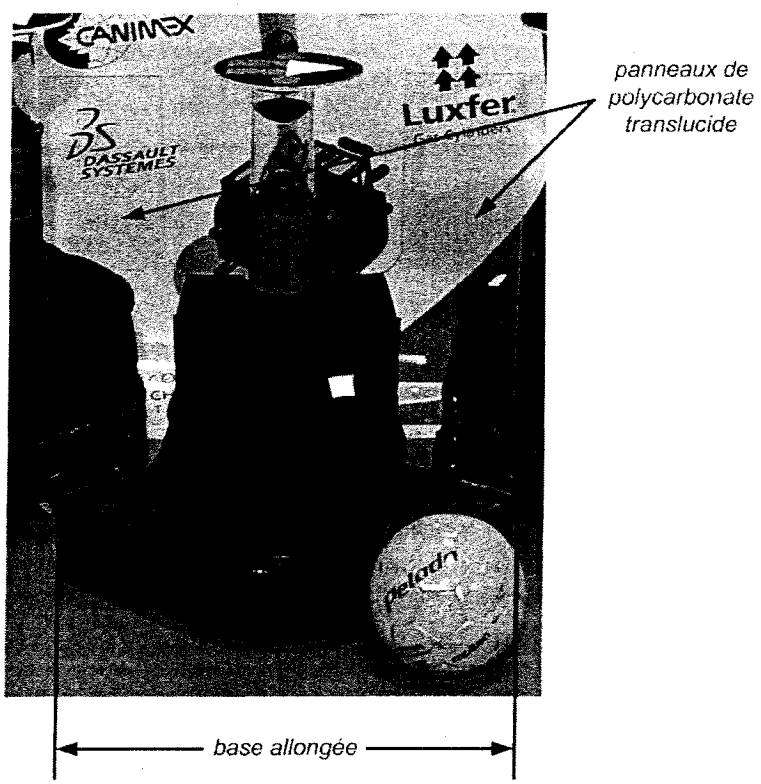


Figure 4.19 : Aspect physique du gardien de but utilisé à la *RoboCup* 2005.

En raison de son aspect différent et des règlements qui stipulent qu'un seul robot de l'équipe doit être spécialement identifié comme étant le gardien de but, un patron spécifique a été développé pour ce joueur et est choisi de façon statique selon l'identité du robot, comparativement aux autres patrons et rôles qui sont choisis de façon dynamique selon la situation de jeu. Ce patron constitue donc en réalité un rôle qui pourrait très bien être inséré dans les autres patrons. Afin de limiter le nombre de nœuds dans les autres patrons, et donc de simplifier les choses, et afin de clarifier l'état particulier de joueur, il a été préféré de le laisser à part dans son propre patron.

La Figure 4.20 présente le détail du patron de jeu *GoalKeeper*. Ce patron constituant en fait un rôle, il est normal que le patron mène directement aux différents comportements permettant au gardien de bien accomplir sont travail. Initialement, le gardien était en mesure d'effectuer son travail avec beaucoup de liberté dans le choix du comportement approprié. En effet, lorsque le ballon était menaçant, le gardien prenait le comportement *ProtectGoal*, mais lors de conditions favorables, il pouvait tenter de prendre possession du ballon et même effectuer des passes et des tirs, comme le démontrent le choix de 5 comportements différents. Par contre, dû à son aspect physique particulier, par exemple l'indisponibilité des botteurs pneumatiques en raison des modules de blocage du ballon, ainsi que pour des raisons de robustesse de son système de vision, le gardien de but a été strictement limité au comportement *ProtectGoal*. En effet, les zones des buts sont des régions particulièrement sensibles pour le système de localisation par vision omnidirectionnelle et des instabilités peuvent survenir lorsque par exemple le gardien s'aventure trop profondément dans son propre but. Ce joueur jouant un rôle particulièrement crucial pour le jeu défensif de l'équipe, l'utilisation de ses comportements est conséquemment limitée à un comportement simple mais assurant un jeu défensif robuste et efficace.

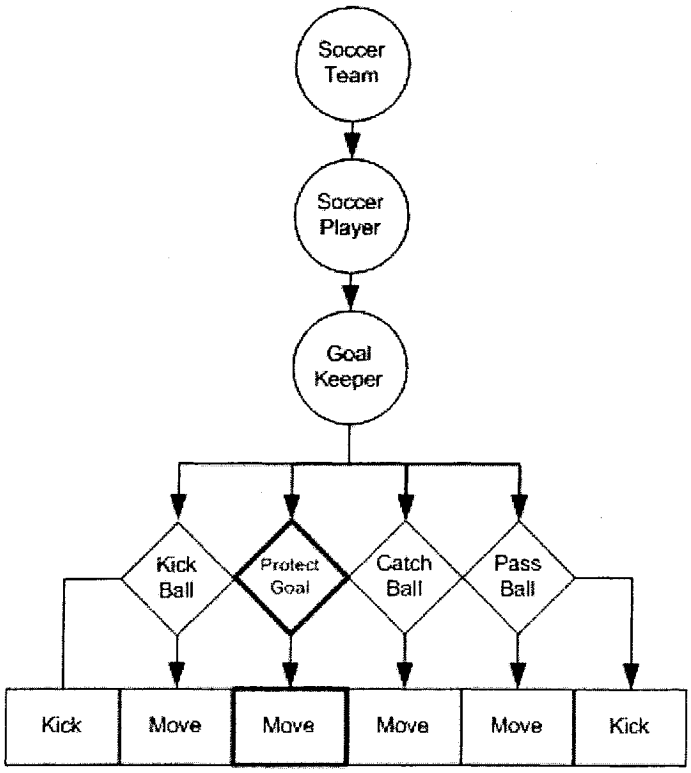


Figure 4.20 : Définition du patron de jeu *GoalKeeper*.

4.4 Mécanisme de supervision décisionnelle

Comme il a été mentionné en 4.2.7, le serveur d'équipe permet entre autre de partager les informations entre les robots et de recevoir les commandes de l'arbitre via le « referee box », mais il permet également d'implanter certains mécanismes de supervision décisionnelle. La supervision décisionnelle développée pour les robots joueurs de soccer aura trois objectifs distincts :

- Répondre adéquatement aux commandes de l'arbitre
- Assurer la sélection d'un patron de jeu commun à tous les coéquipiers (à l'exception du gardien de but) en situation de jeu
- Assurer la sélection d'un rôle distinct pour chacun des coéquipiers (à l'exception du gardien de but) en situation de jeu

Afin d'effectuer son travail, le serveur peut compter sur une information centralisée qui est mise à jour par chacun des robots en utilisant le service *RobotInfoExchange*. Ce service permet donc au robot de transmettre sa position et orientation sur le terrain ainsi que celles du ballon, le tout mesuré par son propre système de perception. Le robot reçoit en retour une matrice contenant les mesures prises par les autres robots.

Le serveur d'équipe doit aussi se charger de recevoir les vecteurs décisionnels des joueurs et de mettre à jour la matrice décisionnelle de l'équipe de joueurs, puis il doit déterminer une matrice de supervision appropriée et transmettre les vecteurs de supervision aux différents robots. Pour implanter ces fonctionnalités, le serveur offre aux robots un service *RobotDecisionExchange* qui permet aux robots de transmettre leurs vecteurs décisionnels respectifs et de recevoir en retour leurs vecteurs de supervision.

Le vecteur décisionnel du robot *i* peut être défini ainsi :

$$V_{D_i} = \begin{bmatrix} M_i & P_i & R_i & C_i \end{bmatrix}$$

Ce vecteur reflète les 4 hiérarchies décisionnelles de la machine *Soccerteam_HDM* : choix du mode (M_i), du patron (P_i), du rôle (R_i) et du comportement (C_i).

La matrice décisionnelle de l'équipe peut donc être définie par :

$$M_D = \begin{bmatrix} M_1 & P_1 & R_1 & C_1 \\ M_2 & P_2 & R_2 & C_2 \\ M_3 & P_3 & R_3 & C_3 \\ M_4 & P_4 & R_4 & C_4 \\ M_5 & P_5 & R_5 & C_5 \end{bmatrix}$$

Le vecteur de supervision et la matrice de supervision sont définis de façon analogue. Sachant que la valeur numérique *–1* sert à n'imposer aucune supervision, la matrice de supervision prendra par défaut la valeur de :

$$M_S = \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}$$

Chacun des robots recevra à ce moment un vecteur de supervision ne contenant que des *–1* et fonctionnera donc en mode non supervisé.

4.4.1 Aspect temporel de la supervision

Étant donnée l'architecture de communication utilisée, chacun des robots communiquera avec le serveur de façon asynchrone. Ainsi, le serveur doit tenir compte de ce fait dans son mécanisme de supervision décisionnelle. Sachant qu'un robot peut appeler le service *RobotDecisionExchange* à n'importe quel instant, il réévalue à chaque appel du service (même service appelé par tous les robots) la nécessité d'imposer une supervision décisionnelle.

Pour ce qui est de la supervision du choix de patron et du rôle une procédure particulière est utilisée. Puisque la communication se fait de façon asynchrone, la détection d'un conflit se doit de tenir compte d'un certain délai permettant au serveur de s'assurer que le système n'est pas simplement en transition et qu'il n'y a pas lieu de superviser. Lors de la détection d'une transition suite à la mise à jour d'un vecteur décisionnel d'un des robots (la matrice décisionnelle change de valeur pour les hiérarchies considérées), le serveur va alors attendre que chacun des autres robots mette à jour son vecteur décisionnel pour stipuler s'il y a situation conflictuelle ou non.

De plus, puisque la situation conflictuelle peut durer un certain temps, le serveur se doit de voir à ce que la supervision décisionnelle se propage dans l'équipe de robots de façon à éliminer le conflit de façon permanente et non ponctuelle. Ainsi, lors de la détection d'un conflit, la supervision sera imposée au système pour une période de temps prédéfinie pour permettre la propagation de la supervision. Durant cette période de supervision, aucun autre conflit ne pourra être traité, mais étant donnée la supervision il ne devrait pas en survenir. La durée de cette période est ajustable. Actuellement, elle a été ajustée grossièrement de façon expérimentale afin d'en limiter la durée pour limiter la perte d'opportunité engendrée, tout en permettant de régler de

façon permanente la majorité des conflits. Un travail plus exhaustif serait nécessaire afin de déterminer une valeur plus précise pour cette durée.

4.4.2 Réponse aux commandes de l'arbitre

Afin de répondre aux commandes de l'arbitre, le superviseur se doit dans un premier temps de détecter les situations qui demandent supervision. Cela se fait grâce au lien de communication entre le serveur d'équipe et le « referee box » (voir 4.2.7). Le superviseur doit en fait constamment superviser le mode des robots puisqu'il est le seul à connaître l'état du jeu reçu du « referee box ». En effet, au départ la machine décisionnelle est au mode *PlayerStop* et pour passer aux modes *PrepareTeam* ou *SoccerPlayer* il faut qu'il se produise un événement au « referee box » et que le serveur d'équipe le transmette à l'équipe.

Par exemple, sachant que le mode *SoccerPlayer* se voit assigner l'identité numérique *0* et le mode *PrepareTeam* l'identité *1*, lorsque le serveur recevra une commande du « referee box » pour lancer le jeu, commande *START*, il pourra imposer aux robots la matrice de supervision suivante :

$$M_S = \begin{bmatrix} 0 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 \end{bmatrix}$$

Cette matrice de supervision permet donc d'imposer le mode *SoccerPlayer* à tous les robots tout en les laissant libre pour le reste de la hiérarchie décisionnelle.

Ensuite, lorsque le serveur recevra des commandes d'arrêts de jeu il devra imposer momentanément une pause suite à l'arrêt en imposant le mode *PlayerStop* dans la première colonne de la matrice de supervision. Et lorsqu'il recevra le type d'arrêt il pourra lancer la préparation de l'équipe en vue de la reprise du jeu. Par exemple, si l'arbitre appelle un coup de pied de coin (« corner ») en faveur de l'équipe, comme ce qui est présenté à la Figure 4.12, et sachant que le patron *CornerKick* se voit attribuer l'identité *6*, la matrice de supervision deviendra :

$$M_S = \begin{bmatrix} 1 & 6 & -1 & -1 \\ 1 & 6 & -1 & -1 \\ 1 & 6 & -1 & -1 \\ 1 & 6 & -1 & -1 \\ 1 & 6 & -1 & -1 \end{bmatrix}$$

À première vue, le mécanisme de supervision décisionnelle semble demander une manipulation de valeurs numériques très peu intuitive. Cependant, le superviseur ainsi que les robots peuvent se baser sur des prédéfinitions d'identités qui éliminent l'utilisation de valeurs numériques qui sont remplacées par des constantes plus intuitives. Par exemple, pour les patrons associés aux arrêts de jeu, la ligne suivante peut être utilisée tant par les robots que par le superviseur pour identifier les différents patrons :

```
enum PREPARE_NODES {KICKOFF, WAITKICKOFF, GOALKICK, WAITGOALKICK, FREEKICK,
WAITFREEKICK, CORNERKICK, WAITCORNERKICK, PENALTY, WAITPENALTY, THROWIN,
WAITTHROWIN, INITPOS, DROPPED_BALL};
```

4.4.3 Supervision du choix de patron

Étant donnée la possibilité de conflits décisionnels au sein de l'équipe pour le choix du patron lorsqu'elle se trouve en mode *SoccerPlayer*, le serveur doit détecter les conflits et les traiter. L'objectif est ici que tous les robots de l'équipe utilisent le même patron de jeu à l'exception du gardien.

Sachant que les patrons se sont vus assigner les identités suivantes : *0* pour *GoalKeeper*, *1* pour *DefenseRC* et *2* pour *OffenseRC*, une matrice décisionnelle présentant un conflit pourrait se présenter ainsi (les X sont des valeurs non considérées par le mécanisme de supervision) :

$$M_D = \begin{bmatrix} 0 & 0 & X & X \\ 0 & 1 & X & X \\ 0 & 1 & X & X \\ 0 & 2 & X & X \\ 0 & 1 & X & X \end{bmatrix}$$

La première ligne décisionnelle correspond au gardien de but et ne sera jamais traitée par le serveur pour la détection de conflit. Pour les 4 autres lignes, il existe un conflit puisqu'un des robots, le 4ème, n'est pas sur le même patron que les autres. Il choisit en effet le patron *OffenseRC* quand les autres sont sur *DefenseRC*.

Pour traiter ce conflit, le serveur utilise la même condition que les robots utilisent pour sélectionner le patron, mais il utilise ses propres données centralisées pour le faire. Dans le cas où il juge que le patron *DefenseRC* est approprié, il imposera la matrice de supervision :

$$M_S = \begin{bmatrix} 0 & -1 & -1 & -1 \\ 0 & 1 & -1 & -1 \\ 0 & 1 & -1 & -1 \\ 0 & 1 & -1 & -1 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

4.4.4 Supervision du choix de rôle

Le choix du rôle au sein des patrons *OffenseRC* et *DefenseRC* effectué de façon distribuée présente également un certain risque de conflit décisionnel. Cette fois l'objectif est de voir à ce que chacun des robots prenne un rôle différent. Le serveur sera donc en mesure de détecter les conflits décisionnels et de les traiter. Il sera cependant en mesure de le faire seulement lorsque tous les robots sont sur le même patron et donc lorsqu'il ne traite pas au même instant un conflit au niveau du choix du patron.

Le traitement des conflits décisionnels pour le choix du rôle est particulièrement important au sein de l'équipe de robots joueurs de soccer puisque ces conflits sont souvent la cause d'obstruction lorsque deux robots tentent de choisir le même rôle.

L'exemple considéré utilisera le patron *OffenseRC* mais le traitement est le même pour le patron *DefenseRC* . Au sein du patron *OffenseRC* il existe 4 rôles distincts qui sont identifiés ainsi : *1* pour *TOffense1*, 2 pour *TOffense2*, 3 pour *TOffense3* et 4 pour *TOffense4*. Une matrice décisionnelle présentant un conflit pourrait par exemple être :

$$M_D = \begin{bmatrix} 0 & 0 & X & X \\ 0 & 2 & 1 & X \\ 0 & 2 & 1 & X \\ 0 & 2 & 3 & X \\ 0 & 2 & 4 & X \end{bmatrix}$$

Dans cette matrice décisionnelle, deux robots, le 2ème et le 3ème, tentent de devenir leader tandis que le rôle *TOffense2* n'est pas assuré. Pour traiter ce genre de conflit, le serveur procède dans un premier temps à l'identification du rôle conflictuel, le rôle *TOffense1* dans le cas présent, puis du rôle manquant, le rôle *TOffense2*. Il détermine ensuite le rôle que devrait choisir chaque robot en utilisant les mêmes conditions que ces derniers mais toujours en se basant sur ses données centralisées. Par exemple, s'il juge que le 3ème robot devrait être leader, il imposera la matrice de supervision suivante :

$$M_S = \begin{bmatrix} 0 & -1 & -1 & -1 \\ 0 & 2 & 2 & -1 \\ 0 & 2 & 1 & -1 \\ 0 & 2 & 3 & -1 \\ 0 & 2 & 4 & -1 \end{bmatrix}$$

4.5 En résumé

Les sections précédentes ont donc présenté les différents éléments constitutifs du système expérimental développé et utilisé dans le cadre de ce projet de maîtrise. Le système est ainsi principalement constitué de plusieurs robots mobiles aux caractéristiques électromécaniques et sensorielles originales, d'une architecture de contrôle distribué, d'une architecture de communication puis finalement d'une architecture décisionnelle avec possibilités de supervision, principal sujet du présent projet. Le système a été développé en suivant les règles de la *RoboCup* « Middle Size Robot League » dans le but de s'y conformer afin de participer à des compétitions internationales.

CHAPITRE 5 - RÉSULTATS OBTENUS ET ANALYSE

5.1 Présentation du système en compétition

L'année 2005 a été la première saison de compétition pour le projet *Robofoot ÉPM*. Pour cette première saison de compétition, l'équipe du projet a décidé de participer à deux compétitions.

5.1.1 GermanOpen 2005, Paderborn, Allemagne

L'édition 2005 du *GermanOpen* constituait la toute première compétition du projet *Robofoot ÉPM*. *Robofoot ÉPM* y était la toute première équipe canadienne à s'y présenter, toutes ligues confondues.

Pour l'équipe du projet *Robofoot ÉPM*, cette participation visait deux objectifs bien précis :

1- Valider que le système multi-robots développé répondait bien aux spécifications imposées par la « Middle Size Robot League ».

2- Acquérir une expérience en compétition permettant de préparer de façon éclairée le système multi-robots en vue de la *RoboCup* 2005.

Il y avait ainsi assez peu d'attente au niveau des résultats, l'équipe étant consciente que le calibre de jeu était élevé, et que l'inexpérience de *Robofoot ÉPM* allait paraître. Cependant, le système se devait de fonctionner adéquatement et les possibilités d'amélioration devaient apparaître clairement.

Le Tableau 5.1 résume les faits saillants de cette première compétition. Le tournoi débutait par une ronde préliminaire où deux groupes de 7 équipes étaient formés aléatoirement à partir des 14 équipes participantes. Chaque équipe jouait ensuite 1 match contre chacune des autres équipes de son groupe. À la suite de cette ronde préliminaire, les 4 meilleures équipes de chaque groupe étaient retenues pour la ronde éliminatoire débutant par des quarts-de-finales.

Compétition	***GermanOpen 2005***
Date, lieu	*8 au 10 avril, Paderborn, Allemagne*
Institut universitaire hôte	*Fraunhofer AIS*

Site web	*http://www.ais.fraunhofer.de/GO/2005/*
Nombre d'équipes	*14*
Nombre de pays représentés	*6*
Nombre de matchs joués	*6*
Nombre de victoires	*0*
Nombre de défaites	*4*
Nombre de matchs nuls	*2*
Nombre de buts pour	*3*
Nombre de buts contre	*18*

Tableau 5.1 : Faits saillants de la compétition *GermanOpen* 2005.

Les résultats, quoique plutôt faibles en apparence, ne sont pas si décevants. En effet, le fonctionnement général du système présenté était adéquat, voire impressionnant pour une toute première participation. Très peu d'équipes ont réussi à marquer des buts dès leurs premiers matchs en compétition! Dans l'ensemble donc, le système multi-robots répondait aux attentes et voici les points positifs notés :

- Architecture multi-robots à prise de décision distribuée fonctionnelle
- Réponse adéquate aux commandes de l'arbitre
- Ajustement du système rapide et facile pour des nouvelles dimensions de terrain
- Fonctionnement général du logiciel adéquat et robuste (aucun « bug » observé)
- Fonctionnement général du matériel adéquat et robuste (aucun bris)

Ce qui ressort principalement de ces remarques est que le fonctionnement général du système est tout à fait adéquat, mais bien évidemment plusieurs points moins appréciés ont été relevés. En voici un résumé :

- Manque flagrant de robustesse du système de perception qui oblige la réinitialisation (sortie et réentrée du robot) fréquente de robots qui perdent leur position sur le terrain. L'environnement visuel peu constant (couleurs et éclairage) explique en partie cette situation.
- Gardien de but présentant un temps de réaction beaucoup trop lent et peu adapté aux tirs aériens.
- Coopération multi-robots difficile principalement dû à l'utilisation de règles préétablies trop peu sélectives et difficiles à respecter en raison de la perception distribuée erratique.

- Comportements cruciaux peu efficaces (protection du but et contrôle du ballon entre autre).

Ces points expliquent entre autre le différentiel de buts de –15. Cette statistique est en fait la seule qui soit vraiment décevante, puisque les équipes en présence étaient toutes beaucoup plus expérimentées. De plus, non seulement elles étaient expérimentées, mais le groupe dans lequel *Robofoot ÉPM* se trouvait était nettement plus fort que le second groupe. En effet, parmi les 7 équipes du groupe, 3 équipes on terminé dans les 4 premières places du tournoi (*Tribots* champions, *CoPS* 3ème place et *Philips RoboCup Team* 4ème). Les résultats des équipes à la *RoboCup* viennent confirmer ce déséquilibre, car 4 équipes (les 4 ayant battu *Robofoot ÉPM* : *CoPS*, *Tribots*, *Philips* et *AIS/BIT*) ont déjà atteint les demi-finales et les quarts-de-finales, contre seulement une seule dans le second groupe : *Minho*, vice-champions du tournoi.

Les deux objectifs pour cette première compétition furent donc atteints et la table était mise pour les préparatifs en vue du vrai test, la *RoboCup*.

5.1.2 RoboCup 2005, Osaka, Japon

L'engouement du peuple japonais pour la robotique est palpable au cours d'un événement comme la *RoboCup*. Plus d'une centaine de milliers de visiteurs sont venus sur le site des compétitions et expositions. Les équipes japonaises dominent le tournoi annuel depuis plusieurs années, talonnées de près cependant par d'excellentes équipes allemandes. Il y a aussi plusieurs autres excellentes équipes, néerlandaises, portugaises, chinoises et iraniennes.

Pour *Robofoot ÉPM*, les objectifs étaient bien différents de la première compétition en Allemagne :

1- Se classer parmi les meilleures équipes du tournoi tant dans les matchs que dans les challenges techniques.
2- Démontrer le potentiel du système versatile, modulaire et robuste.
3- Établir des échanges scientifiques avec les équipes présentes.

Afin d'atteindre ces objectifs, des éléments clés avaient été identifiés suite au *GermanOpen* :

- Rendre le système de perception beaucoup plus robuste en général et en particulier pour le gardien de but (problème près des buts).

- Adapter le gardien de but aux tirs aériens et améliorer grandement son temps de réponse.
- Revoir l'efficacité de tous les patrons associés aux commandes de l'arbitre.
- Revoir l'efficacité des comportements associées au contrôle du ballon (*CatchBall*, *DribbleBall* et *KickBall*).
- Améliorer le jeu coopératif par l'utilisation de règles préétablies moins sensibles aux imprécisions de la perception distribuée et par l'implantation de mécanismes de supervision décisionnelle.

Voici donc les éléments clés qui ont guidé les développements en vue de cette compétition. Quatre de ces cinq éléments ont pu être développés grâce à la machine décisionnelle déjà développée. Cette machine a donc été révisée et la version du système documentée au chapitre précédent correspond à la version finale présentée à la *RoboCup* 2005. En effet, on y retrouve les mécanismes de supervision décisionnelle ainsi que l'aspect physique spécifique du gardien, éléments développés entre les deux compétitions.

Afin de faire ressortir les principales statistiques de ce tournoi, le Tableau 5.2 a été dressé. On remarque d'emblée l'envergure considérable du tournoi avec 20 équipes présentes représentant un total de 12 pays, *Robofoot ÉPM* étant les seuls représentants en provenance du continent américain.

Compétition	***RoboCup 2005***
Date, lieu	*13 au 19 juillet, Osaka, Japon*
Institut universitaire hôte	*Université d'Osaka*
Site web	*http://robocup2005.org*
Nombre d'équipes	*20*
Nombre de pays représentés	*12*
Nombre de matchs joués	*6*
Nombre de victoires	*3*
Nombre de défaites	*3*
Nombre de matchs nuls	*0*
Nombre de buts pour	*7*
Nombre de buts contre	*7*

Tableau 5.2 : Faits saillants de la compétition *RoboCup* 2005.

Le format du tournoi débutait par un premier tour de 4 groupes de 5 équipes établis au hasard. Chaque équipe jouait un match contre chacune des équipes de son groupe. *Robofoot ÉPM* y a remporté 3 victoires contre une seule défaite, subie face à *Eigen*, de l'université de Keio au Japon. Le match contre *Eigen*, le tout premier de l'équipe en Coupe du Monde, présentait un score de 0-0 à la mi-temps. Une victoire aurait été un exploit, cette équipe est invaincue depuis deux ans! Malheureusement, un bris de moteur ayant obligé le retrait d'un joueur de *Robofoot ÉPM* ainsi qu'un jeu offensif amélioré de la part de *Eigen* pour la deuxième demie a mené à un score final de 4-0 en faveur de *Eigen*. Trois victoires ont tout de même suivi ce premier match enlevant!

Un deuxième tour dans le but d'identifier les 8 meilleures équipes du tournoi avait lieu par la suite. Les trois premières équipes de chaque groupe accédaient à ce deuxième tour et étaient regroupées selon un ordre préétabli, chaque groupe comportant une première, une deuxième et une troisième position. *Robofoot ÉPM* a été placée avec *WinKIT*, du Kanazawa Institute of Technology du Japon, équipe championne en 2003 et vice-championne en 2004, ainsi qu'avec *Minho*, de l'université de Minho au Portugal, respectable 4ème place en 2005. En raison d'un jeu offensif déficient et de difficultés à s'ajuster sur un nouveau terrain présentant un éclairage plus intense (ballon difficile à localiser par le système de vision des robots), *Robofoot ÉPM* a dû s'incliner respectivement 1-0 et 2-0 face à ces équipes. Pour sa première Coupe du Monde, *Robofoot ÉPM* terminait donc 9ème à égalité avec trois autres équipes. Avec quelques ajustements supplémentaires l'équipe aurait peut-être pû accéder à la ronde éliminatoire ce qui aurait été un exploit pour une nouvelle équipe. Fait à relater, *Robofoot ÉPM* est une des seules équipes de l'histoire à avoir accédé au deuxième tour de qualification dès sa toute première participation.

Les scores serrés de la deuxième ronde ont démontré deux caractéristiques du système développé :

- Jeu défensif efficace : jeu de position bien adapté et robuste tout au long du tournoi.
- Comportements individuels offensifs peu efficaces : vision du ballon difficile avec éclairage très intense et plate-forme à vitesses différentielles désuète pour le contrôle du ballon.

En plus d'un tournoi éliminatoire, la *RoboCup* présente également des challenges techniques au cours desquels les équipes doivent démontrer différentes capacités originales en termes de

développements techniques. Ces challenges sont d'ailleurs une façon d'encourager les équipes à travailler non seulement au développement d'un système très compétitif, mais également de se concentrer sur divers aspects scientifiques reliées au domaine des systèmes multi-robots autonomes.

Pour l'édition 2005, deux challenges étaient prévus : le challenge d'évitement d'obstacles ainsi que le challenge libre. Pour le challenge d'évitement d'obstacles, des membres du comité techniques disposent sur le terrain des boîtes noires, des obstacles, selon trois patrons préétablis. Pour chacun des patrons, un robot de l'équipe évaluée doit tenter de prendre possession du ballon disposé devant lui et de l'entrer dans le but adverse sans toucher aux obstacles. Le pointage est établi en fonction des buts marqués, des contacts avec les obstacles (points négatifs) ainsi qu'en fonction du temps requis pour marquer. Dans ce challenge, *Robofoot ÉPM* s'est méritée une respectable 4ème place grâce à une navigation avec évitement d'obstacles efficace. Pour le second challenge, les équipes devaient présenter un développement original particulièrement intéressant en termes d'ingénierie. Par exemple, des développements en mécanique, en vision ou en intelligence artificielles étaient présentés par certaines équipes. *Robofoot ÉPM* a décidé de présenter les aspects les plus intéressants de la machine décisionnelle développée, c'est-à-dire sa capacité à assigner dynamiquement des patrons de jeu (démonstration des patrons offensifs et défensifs) et des rôles (échange de rôle en bougeant le ballon entre les joueurs) ainsi que la possibilité de faire des passes entre les joueurs, ce que très peu d'équipes sont en mesure de réussir. Ce challenge était noté par les différents chefs d'équipes (« team leader ») et le cumulatif de toutes les évaluations déterminait le classement des équipes. Malgré une 7ème place respectable, *Robofoot ÉPM* anticipait un meilleur résultat pour ce challenge. La difficulté à démontrer sur le terrain les éléments conceptuels concernés (pas de support visuel pour accompagner la présentation) ainsi que la réputation établie de certaines équipes au niveau technique ont probablement mené à ce résultat. Néanmoins, *Robofoot* a obtenu un total de 116 points tandis que les 6ème et 5ème places ont respectivement obtenu 117 et 118,5 points, ce qui est très rapproché comme score. La première position, *Minho*, a obtenu 133 points.

Mis à part tous ces aspects compétitifs, la participation à un tel événement se doit d'être une expérience très riche sur le plan technique et scientifique. De nombreux échanges constructifs ont été réalisés avec différents membres de l'organisation de l'événement et des équipes présentes.

Malgré certains comportements individuels inefficaces contre les meilleures équipes, il n'y a que très peu de points négatifs émanant des résultats du tournoi. L'amélioration par rapport aux performances démontrées au *GermanOpen* est d'ailleurs remarquable. La modularité et la polyvalence de la machine décisionnelle développée ont été mises en évidence. Cette machine a pû être révisée, améliorée à divers niveaux, et ce avec beaucoup de facilité et sans pour autant affecter son fonctionnement global.

5.2 Mesures de performance

5.2.1 Utilisation des ressources système

Pour la simulation un ordinateur de bureau HP Compaq DC7100 a été utilisé. Voici les principales caractéristiques de cet ordinateur.

Composante	*Paramètres*
Système d'exploitation	• Microsoft Windows XP Professional SP2
Compilateur	• Microsoft Visual C++ 6.0
Processeur	• Intel Pentium 4 à 3,20GHz avec « Hyper-Threading »
Chipset	• Intel 945G
Mémoire vive	• 1Go de mémoire SDRAM DDR2 à 667MHz
Disque dur	• SATA de 80Go (1,5Go/s, 7200rpm)

Tableau 5.3 : Caractéristiques de l'ordinateur utilisé pour les simulations.

La simulation du système en entier implique donc l'exécution de l'ensemble des modules logiciels nécessaires ainsi que 10 instances du programme des robots pour simuler un match à 5 contre 5. En simulation, la période du « thread » principal au sein du module *player* est de 50ms. Le tableau suivant résume les résultats qui ont été observés grâce entre autre au *Task Manager* fourni avec *Windows*.

Module logiciel	*Utilisation moyenne CPU*	*Utilisation Mémoire*	*Nombre de « threads »*
Joueurs (*player*, x10)	2%	13,36Mo	7
Serveur (*team_server*)	13%	3,45Mo	15
Simulateur terrain (*soccerfield_sim*)	1,5%	3,15Mo	3
Interface (*Interface*)	0,5%	5,80Mo	2
Visualisateur (*LookatSoccer*)	9%	16,70Mo	1

Tableau 5.4 : Mesures d'utilisation des ressources en simulation.

Chaque joueur simulé demande donc près de 2% du processeur et 13,36Mo de mémoire vive. L'utilisation du CPU, même si elle inclut beaucoup plus que le processus décisionnel (calcul de localisation, communication, contrôle de bas niveau), est très modérée et ce résultat est

intéressant dans le but de laisser le processeur disponible pour des opérations de traitement complexes comme ceux demandés par la vision artificielle. Cependant, l'utilisation de la mémoire vive est relativement importante. De ces 13,36Mo, environ 6,2Mo sont dûs à la machine décisionnelle. Ceci s'explique par le grand nombre de nœuds présents dans la machine (128 si on inclue les actions). Certains moyens pourraient être pris pour réduire la mémoire utilisée. Si par exemple on applique le concept de singleton pour éviter de dupliquer un nœud utilisé à plusieurs endroits dans la machine, l'utilisation de mémoire serait considérablement réduite, mais il serait alors impossible de définir un contexte différent pour chaque nœud, ce qui contraindrait la liberté décisionnelle.

Avec une utilisation totale de près 45% du CPU pour une simulation de 10 robots avec tous les outils nécessaires comme le visualisateur et l'interface. Les mesures obtenues permettent de démontrer la faisabilité de la simulation complète du système sur un seul ordinateur. Cependant, compte tenu de la puissance de l'ordinateur utilisé, on constate que le tout est relativement gourmand. Par exemple, le système est également simulé sur un ordinateur portable avec processeur Celeron 1,2GHz et 512Mo de RAM et sur cette machine l'ensemble des composantes consomment près de 100% du CPU.

Pour les essais sur le système réel, il était intéressant d'analyser l'utilisation des ressources de la machine décisionnelle au sein d'un des robots du système. L'ordinateur utilisé sur les robots joueurs de soccer est un Kontron VIPer830. Ses principales caractéristiques sont résumées ci-dessous.

Composante	***Paramètres***
Système d'exploitation	• Debian Linux noyau 2.4.18 modifié
Compilateur	• gcc (version 2.9.5)
Processeur	• Intel Pentium III Low Power à 800MHz (ou Celeron à 566MHz)
Chipset	• Intel 440BX
Mémoire vive	• 256Mo de mémoire SDRAM à 100MHz (ou 66MHz)
Disque dur	• Microdrive ATA de 1Go (33Mo/s, 3600rpm)

Tableau 5.5 : Caractéristiques de l'ordinateur embarqué des robots joueurs de soccer.

Ces ordinateurs sont disponibles avec deux versions différentes de processeurs, un robot du système étant muni du processeur *Pentium III* et les 5 autres étant munis de *Celeron* qui consomment moins d'énergie mais qui offrent une puissance de calcul plus faible. Pour les essais sur le système réel, chacun des robots est pourvu d'un « thread » principal roulant périodiquement

avec une période de 20ms. Pour prendre des mesures, l'utilitaire *top* disponible sur toutes les distributions de *Linux* a été utilisé. Comme cet utilitaire donne des informations plus détaillées que le *Task Manager* de *Windows*, il a été possible d'étudier plus précisément le comportement du logiciel, entre autre pour bien décortiquer ce qui concerne les traitements de vision et ceux du processus cognitif.

Pour ce qui est de l'utilisation du processeur, il n'est pas surprenant de constater que le traitement inévitablement gourmand demandé par le système de vision requiert la majorité du processeur. En effet, sur le processeur *Pentium III* ce traitement requiert entre 40 et 65% du processeur, pour une moyenne d'environ 51%. Cette grande variabilité s'explique par le fait que le traitement s'adapte aux positions et dimensions des objets présents dans l'image en établissant dynamiquement des zones de recherche pour chaque objet. De plus, le nombre d'objets observés, entre autre les obstacles présents à proximité, peut varier et donc influencer grandement le temps de traitement nécessaire. Sur le processeur *Celeron*, le traitement du système de vision demande typiquement entre 60 et 70% avec une moyenne d'environ 64%. Il est intéressant de constater que ce traitement ne demande pas 100% du processeur, ce qui permet au système de vision de traiter le maximum d'images disponibles et au système d'exploitation d'allouer suffisamment de temps CPU à d'autres tâches comme le processus décisionnel.

Il est important de constater que pour le processus décisionnel les ressources sollicitées peuvent aussi varier considérablement, puisque d'une ligne décisionnelle à une autre le nombre de niveaux hiérarchiques peut changer ainsi que la complexité des traitements nécessaires. En considérant tout ce que le module *Brain* doit effectuer pour fonctionner en mode joueur de soccer (communication, traitement d'information, processus décisionnel), sa consommation à l'arrêt (mode *PlayerStop*). se situe entre 4,6% et 5,2% sur le *Pentium III* et entre 6,5% et 7,6% sur le *Celeron*. Lorsque le robot est en mode joueur (mode *SoccerPlayer*), le module *Brain* peut demander plus de ressources puisqu'il y a plus de traitement nécessaire (plus de décision à prendre, plus d'information à traiter). Sur le processeur *Pentium ///*, le processus décisionnel peut demander jusqu'à 7,5% dans ce mode et jusqu'à 10,5% sur le *Celeron*. Cette augmentation est cependant sporadique et en moyenne la consommation se rapproche de celle obtenue à l'arrêt, soit environ 5,6% sur le *Pentium III* et 7,8% sur le *Celeron*. À la lumière de ces résultats, la consommation CPU moyenne du processus décisionnel peut être considérée faible relativement

aux autres traitements nécessaires au fonctionnement du robot. D'ailleurs, avec une utilisation totale moyenne du processeur d'environ 56% sur le *Pentium III* et de 71% sur le *Celeron*, il reste beaucoup de puissance de calcul disponible pour des développements futurs.

Les robots étant munis de 256Mo de mémoire RAM, il est nécessaire d'en vérifier l'utilisation par le programme des robots joueurs de soccer afin de ne pas dépasser cette quantité. Il été mesuré que le programme des robots demande 4,4% de la mémoire totale du système, soit environ 11,26Mo. De cette quantité, il y en a 0,5% qui est prise pour le système de vision (1,28Mo). Les modules *Brain* et *Controller* nécessitent donc dans l'ensemble près de 10Mo de mémoire et la machine décisionnelle demande près de 4,7Mo. Ces valeurs sont du même ordre de grandeur que celles prises en simulation sur *Windows* mais légèrement inférieures. Malgré une consommation importante de mémoire, cette consommation demeure faible en proportion de la mémoire totale disponible.

5.2.2 Mesures temporelles

Étant donnée l'application temps réel visée, il est également intéressant d'étudier le temps demandé pour l'exécution du processus décisionnel. Avec la classe *Sandglass* de la librairie *MICROB*, le temps pris pour l'exécution a été mesuré de la façon suivante :

```
// Initialisation de l'horloge
braintimer.delta();

// Processus décisionnel
hdm.set_robot_info(robot);
hdm.process();

// Mesure du temps écoulé
HDMtime = braintimer.delta();
```

Figure 5.1 : Mesure de la durée du processus décisionnel.

La méthode *Sandglass::delta()* retourne le temps écoulé, en s, depuis le dernier appel de la fonction. Le processus décisionnel est donc délimité par deux appels à cette méthode.

Lorsque le robot n'a pas débuté le match, il se retrouve en mode *PlayerStop*. Dans ce mode, représenté à la Figure 4.11, le mode est directement un comportement et donc la ligne décisionnelle est très courte, des temps relativement courts sont donc attendus. Ensuite, pour le mode *SoccerPlayer*, le temps requis dépend de la ligne décisionnelle adoptée. Par exemple, le gardien de but, qui choisit le patron *GoalKeeper* (voir Figure 4.20), possède un étage décisionnel

de moins. De plus, l'évitement d'obstacles (au sein de *MotionControl*), processus gourmand en traitement, est désactivé pour ce joueur. Pour les autres patrons de jeu, la ligne décisionnelle influencera également le temps requis pour compléter le processus puisque dépendamment du rôle joué par un robot, il peut avoir plus ou moins de traitement à faire. Afin d'observer l'influence des lignes décisionnelles sur le temps requis, des mesures ont été prises pour le mode *PlayerStop*, ainsi que pour les patrons *GoalKeeper* et *OffenseRC* (voir Figure 4.15) du mode *SoccerPlayer*. Le tableau qui suit présente les temps mesurés en simulation.

Ligne décisionnelle	***Temps min. (us)***	***Temps max. (us)***	***Moyenne (us)***
PlayerStop	31,5	45,1	38,6
SoccerPlayer->GoalKeeper	17,8	22,5	20,5
SoccerPlayer->OffenseRC->TOffense1	113	245	195
SoccerPlayer->OffenseRC->TOffense2-3-4	43,5	62,9	56,7

Tableau 5.6 : Durée du processus décisionnel mesurée en simulation.

Les temps mesurés sont très courts ce qui démontre en quelque sorte l'efficacité du mécanisme décisionnel. On remarque cependant que ce temps peut varier considérablement en fonction de la ligne décisionnelle adoptée par le robot. Lorsque le robot joue le rôle *TOffense1* la durée du processus décisionnel devient très longue comparativement à celle des autres lignes décisionnelles. Il est clair que le mécanisme décisionnel ne peut être déterministe car la durée totale d'une ligne décisionnelle est difficilement prédictible et cette durée peut varier selon les différentes lignes possibles. Il est tout de même possible d'évaluer la durée de la ligne la plus lente d'une machine donnée et donc d'évaluer sa durée maximale d'exécution, information pertinente pour un système déterministe, comme celui des robots joueurs de soccer.

Sur les robots, la durée du processus décisionnel mesurée est beaucoup plus grande. Les mesures prises lors des essais sont résumées dans le tableau qui suit. On remarque que les mesures ne sont pas différenciées pour les différentes lignes décisionnelles comme il a été mesurée en simulation. En fait, les mesures ont démontré que sur le système réel la durée du processus était très peu influencée par la ligne décisionnelle adoptée. En effet, les mesures présentées dans le tableau sont valides peu importe la ligne décisionnelle adoptée par le robot. Il est toutefois difficile d'expliquer ce résultat puisqu'une variabilité, semblable à celle mesurée en simulation, était attendue. Néanmoins, les mesures observées permettent de confirmer l'efficacité du mécanisme décisionnel ainsi que la compatibilité de la machine développée avec le logiciel temps-réel

déterministe des robots joueurs de soccer. Avec une durée total de près de 0,1ms, ceci laisse amplement de marge de manœuvre afin de respecter la période de 20ms du « thread » principal.

Ordinateur embarqué	*Temps min. (us)*	*Temps max. (us)*	*Moyenne (us)*
VIPer 830 avec processeur Pentium III LP à 800MHz	107	135	120
VIPer 830 avec processeur Celeron à 566MHz	145	172	158

Tableau 5.7 : Durée du processus décisionnel mesurée sur les robots.

Il y a donc effectivement une bonne différence de performance entre l'ordinateur utilisé en simulation et celui utilisé sur les robots. Typiquement, selon les données du manufacturier *Intel*, l'ordinateur utilisé en simulation serait près de 6 fois plus puissant que l'ordinateur des robots muni du processeur *Pentium III*. Ce qui expliquerait donc la différence au niveau des mesures.

5.3 Conflits décisionnels et supervision décisionnelle

Comme il a été mentionné à quelques reprises, un système multi-robots à perception et cognition distribuées peut être inévitablement la source de conflits décisionnels. Dû à une certaine imprécision des systèmes de perception, chacun des robots du système devra baser sa prise de décision sur une information de l'environnement qui diffère d'un robot à l'autre. Si la prise décision se fait en considérant des objets de l'environnement commun à chacun des robots du système, les conflits peuvent parfois inévitables. Par exemple, pour les robots joueurs de soccer, la plupart des décisions sont basées sur la position relative des joueurs et du ballon sur le terrain. Malgré des règles décisionnelles révisées ayant comme objectif de réduire les sources de conflits décisionnels en assurant une meilleure tolérance aux imprécisions, les conflits demeurent inévitables.

Pour le démontrer, des mesures ont été prises sur le système en fonction. La Figure 5.6 présente l'assignation dynamique des rôles au sein du patron *OffenseRC* pour une période deux minutes. Il est intéressant de constater l'aspect dynamique des machines décisionnelles distribuées qui présentent des changements de rôle très fréquents. Cependant, comme il est souhaité d'avoir chacun des coéquipiers dans un rôle distinct, de nombreux conflits décisionnels sont observés. Les conflits associés à cette période de mesure de deux minutes sont présentés à la Figure 5.7. Une valeur de 1 signifie qu'il y a conflit et une valeur de 0 signifie que les 4 rôles sont comblés

adéquatement. Selon ces résultats, le système présente un conflit décisionnel pour une portion de 48,11% de la durée totale de l'essai. Cette valeur est évidemment gênante par son importance, malgré un fonctionnement général du système beaucoup moins erratique que le laisse croire cette valeur.

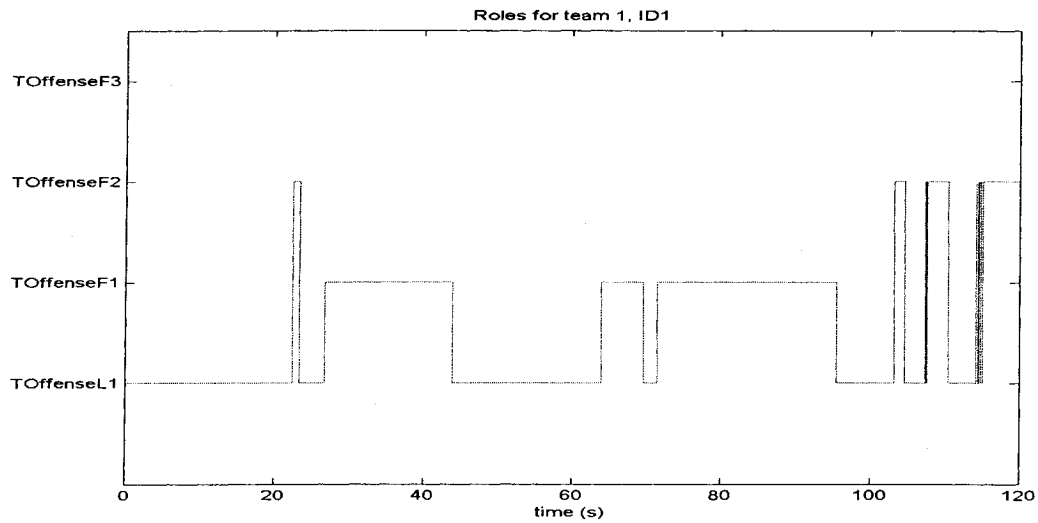


Figure 5.2 : Attribution du rôle du joueur 1 pour l'équipe sans supervision décisionnelle.

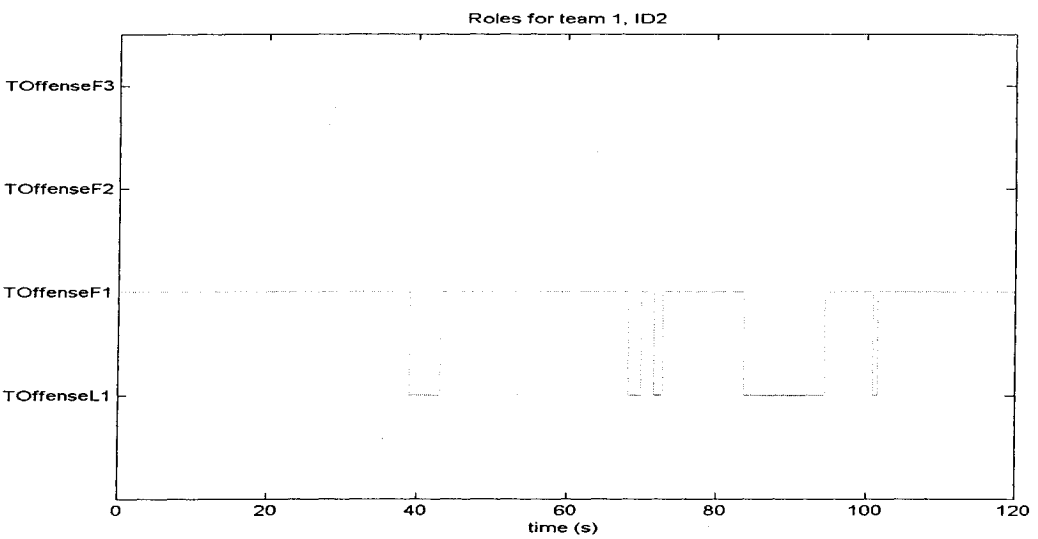


Figure 5.3 : Attribution du rôle du joueur 2 pour l'équipe sans supervision décisionnelle.

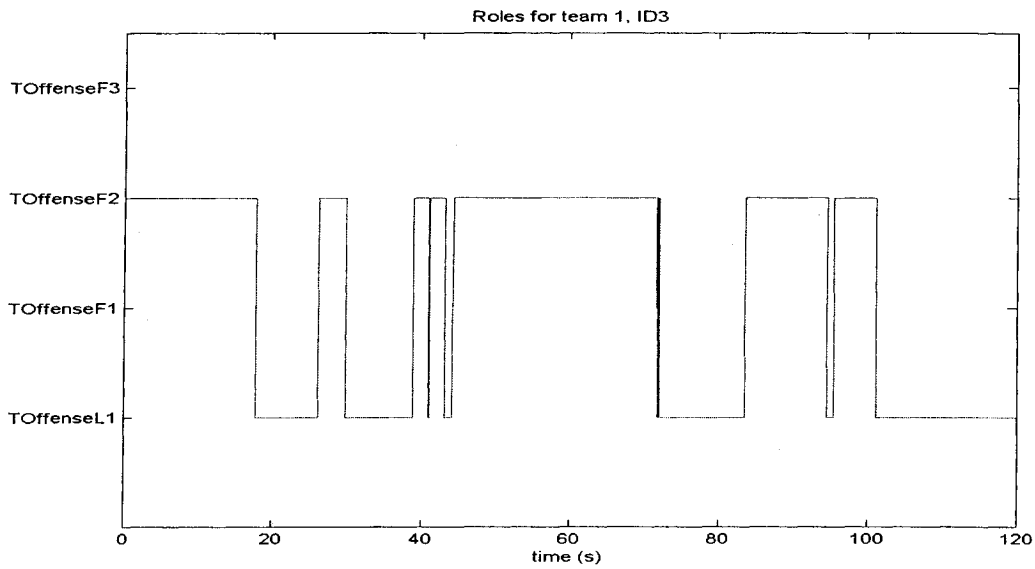


Figure 5.4 : Attribution du rôle du joueur 3 pour l'équipe sans supervision décisionnelle.

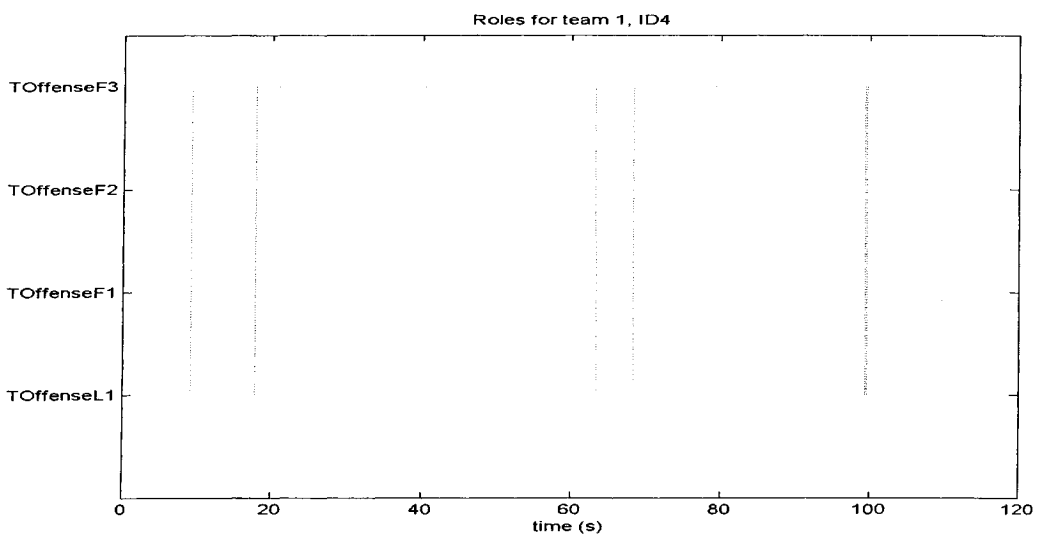


Figure 5.5 : Attribution du rôle du joueur 4 pour l'équipe sans supervision décisionnelle.

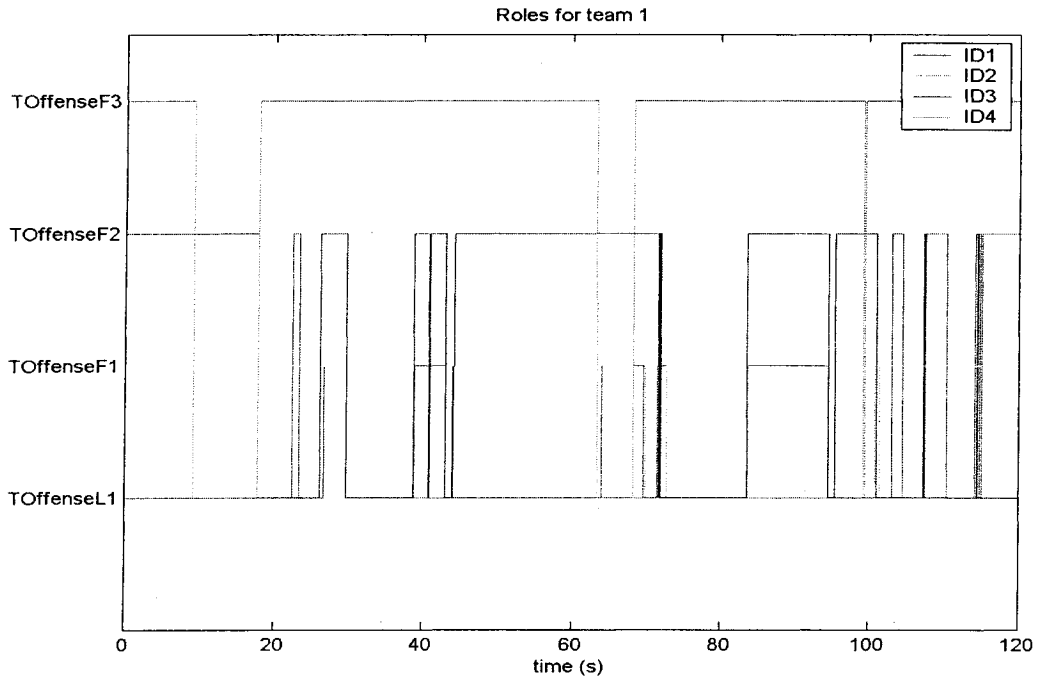


Figure 5.6 : Attribution des rôles pour l'équipe de joueurs de soccer sans supervision décisionnelle.

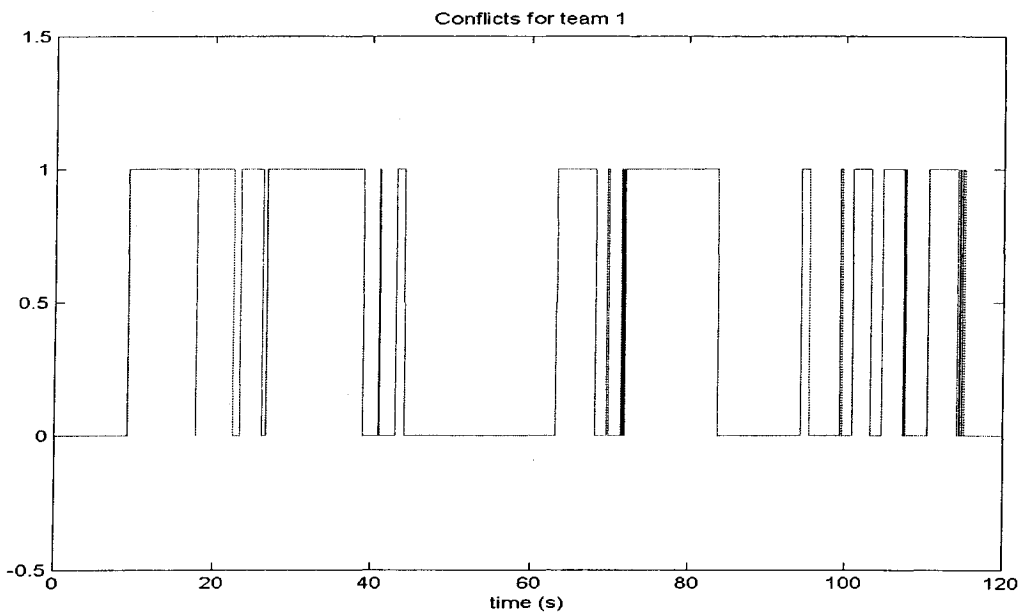


Figure 5.7 : Conflits pour l'équipe de joueurs de soccer sans supervision.

Malgré un fonctionnement en apparence acceptable lorsque le système est observé sur le terrain de jeu, ces conflits sont indésirables. Lorsqu'ils surviennent au moment où l'équipe tente de reprendre le contrôle du ballon (sur le rôle *TOffense1*, conflit le plus fréquent), cela peut se traduire par une perte d'efficacité considérable, un robot seul ayant à ce moment une bien plus grande probabilité de prendre possession du ballon! De plus, étant donnée la plate-forme motrice à contrainte non-holonome et une méthode d'évitement d'obstacles qui se doit de tolérer les obstacles à proximité, les robots se coincent fréquemment les uns contre les autres lors des conflits.

Ceci étant dit, il est souhaitable de chercher à éliminer les conflits décisionnels et le moyen proposé dans la présente architecture est la supervision décisionnelle par un serveur central. Les mécanismes décrits en 4.4 ont donc été implantés dans le but de réduire considérablement les conflits. Le résultat est assez convaincant si l'on se fie aux mesures prises sur le système. Les figures ci-dessous présentent l'assignation dynamique des rôles, toujours pour le patron *OffenseRC* et pour une durée de deux minutes, mais cette fois avec la supervision décisionnelle activée. On remarque que les conflits ont considérablement diminué puisqu'ils sont rapidement observés et traités par le superviseur. La proportion de conflits pour cet essai se chiffre à 2,39% de la durée totale, ce qui correspond principalement au temps de propagation de la supervision au sein de l'équipe. Malgré la supervision décisionnelle qui peut parfois empêcher un robot du système de prendre un certain rôle, on remarque que ce mécanisme n'est pas contraignant au niveau du dynamisme du système puisque les échanges de rôle demeurent très fréquents.

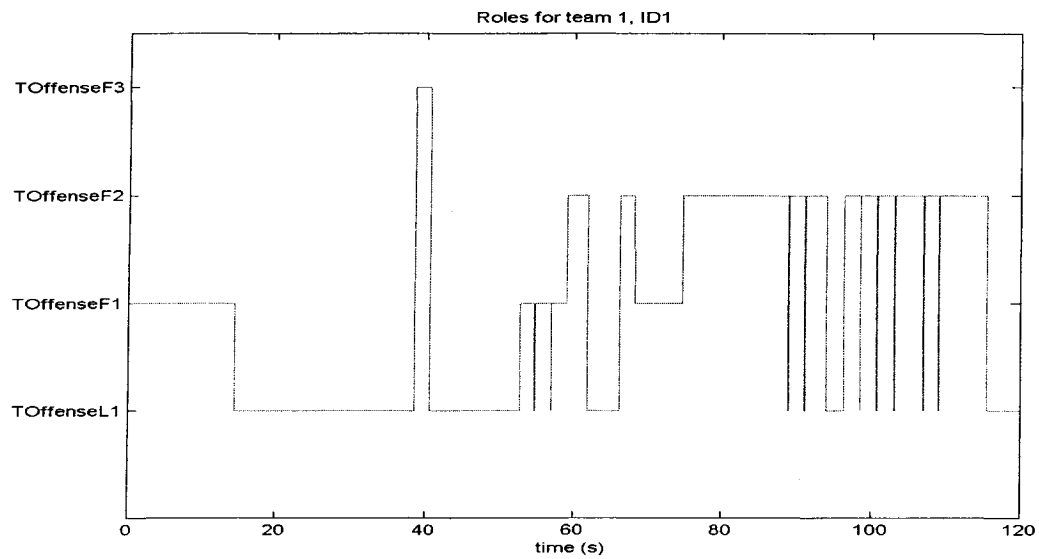


Figure 5.8: Attribution du rôle du joueur 1 pour l'équipe avec supervision décisionnelle.

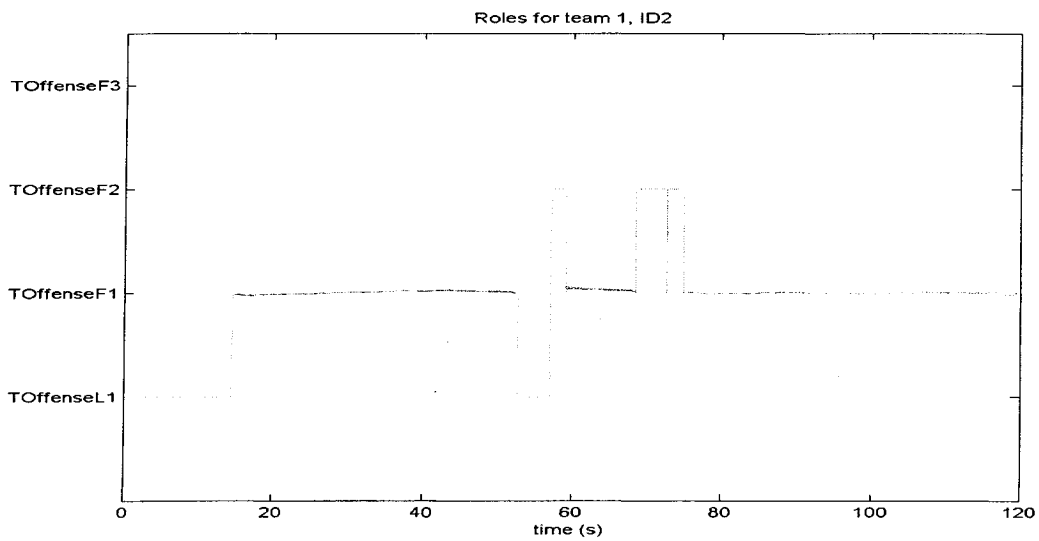


Figure 5.9: Attribution du rôle du joueur 2 pour l'équipe avec supervision décisionnelle.

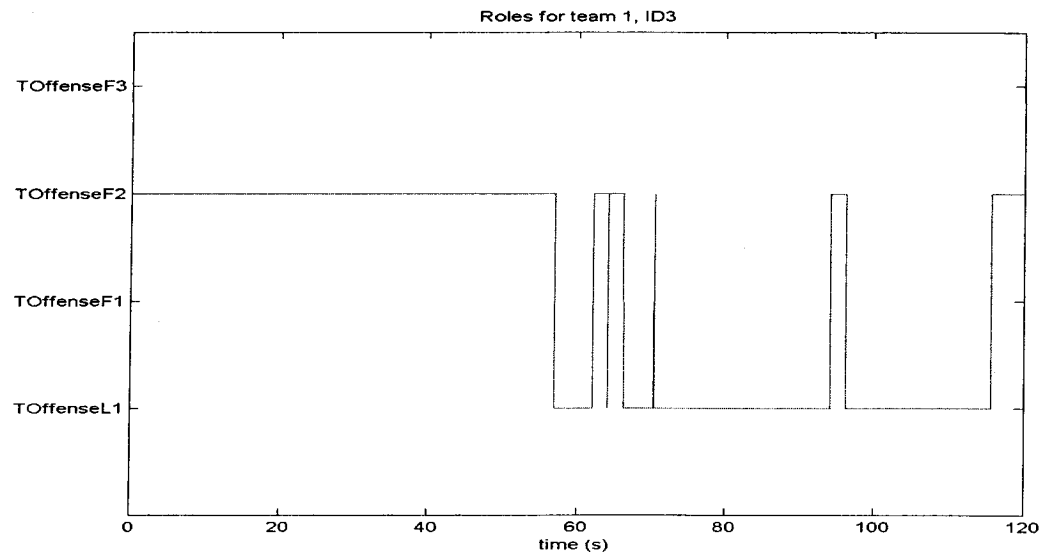


Figure 5.10: Attribution du rôle du joueur 3 pour l'équipe avec supervision décisionnelle.

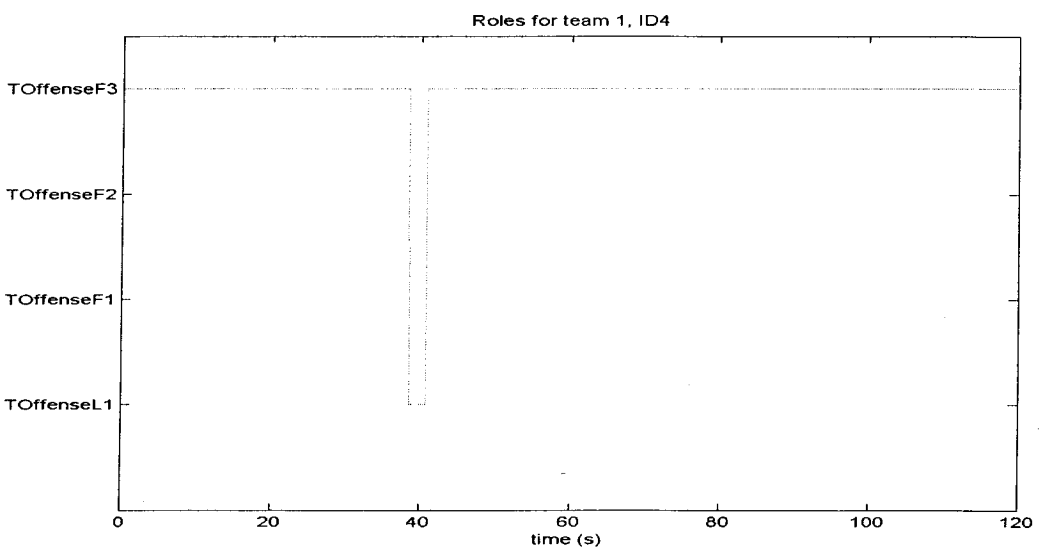


Figure 5.11: Attribution du rôle du joueur 4 pour l'équipe avec supervision décisionnelle.

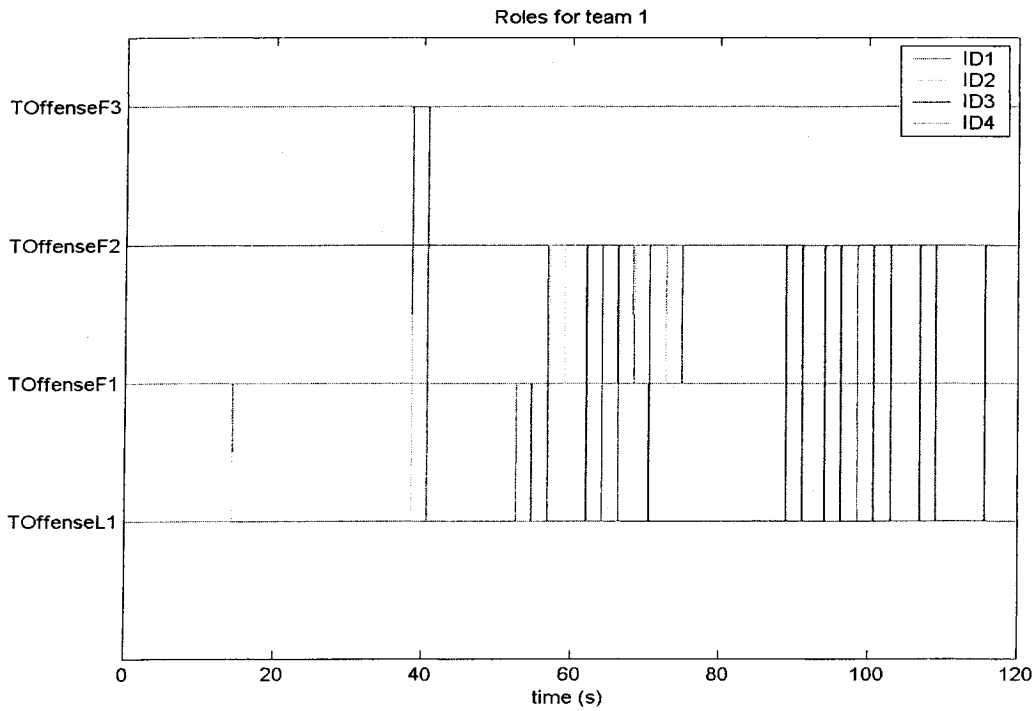


Figure 5.12 : Attribution des rôles pour l'équipe de joueurs de soccer avec supervision décisionnelle.

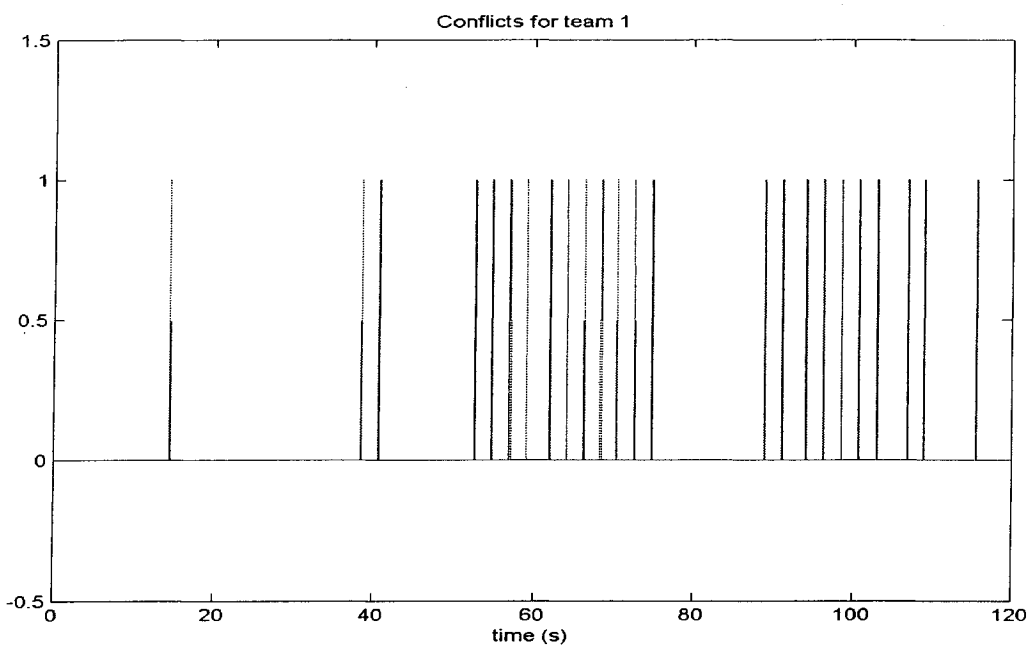


Figure 5.13 : Conflits pour l'équipe de joueurs de soccer avec supervision.

5.4 Respect des spécifications

Au chapitre 3 les différents objectifs visés par le développement de la machine décisionnelle ont été établis ainsi qu'une liste de spécifications fonctionnelles. Ces éléments étaient des critères de conception et il était important de les respecter.

Au sens large, le but du projet était d'établir certaines bases pour le développement de systèmes multi-robots et coopératifs. Ce but a été atteint grâce à une recherche approfondie au cœur du domaine des architectures pour de tels systèmes, puis à la définition d'une machine décisionnelle étant capable de répondre à un vaste éventail d'applications et offrant des possibilités d'implémentation très souples. Les deux objectifs du projet ont également été atteints puisqu'une architecture décisionnelle pour systèmes multi-robots avec mécanismes spécifiquement conçus pour aider à la coopération a bel et bien été conçue et elle a été validée par une implantation sur les robots joueurs de soccer développés à l'École Polytechnique spécifiquement pour la « Middle Size Robot League » de la *RoboCup*, robots qui ont été présentés deux fois en compétition internationale.

Plus précisément, cette machine décisionnelle se devait de répondre adéquatement aux nombreuses spécifications fonctionnelles. Premièrement, l'architecture développée se devait d'être adéquate pour un large éventail d'applications. Malgré qu'elle ait seulement été testée sur des robots joueurs de soccer, il est possible d'imaginer toutes sortes d'applications utilisant ce genre d'architecture : robots d'exploration spatiale, système robotisé d'agriculture, cour d'entreposage robotisé, etc. L'architecture est également en mesure de gérer un groupe de robots, qu'il soit homogène ou hétérogène. Pour un système homogène, des machines décisionnelles identiques peuvent être distribuées sur les robots du système. Pour un groupe hétérogène, les machines peuvent identiques et la prise de décision et l'activation peuvent être spécifiques selon le type de robot concerné. La supervision décisionnelle est tout aussi possible dans ce cas. On a vu que la communication peut être d'une grande utilité lorsque des mécanismes de supervision sont souhaités. L'architecture est donc en mesure de bénéficier de la communication explicite lorsqu'elle est possible. La machine décisionnelle hiérarchique a cependant été conçue pour fonctionner adéquatement de façon distribuée sans communication, tel que le demandaient les spécifications. Les spécifications demandaient également à ce que le nombre de hiérarchies d'une

machine soit arbitraire et que la supervision puisse s'effectuer sur toute la hiérarchie. Ceci est possible grâce aux concepts que de ligne décisionnelle et de matrice décisionnelle pouvant s'établir dynamiquement. Les fonctions décisionnelles se devaient également d'être arbitraires et elles le sont, une telle fonction n'étant que la sélection d'un nœud et le mécanisme de sélection est tout à fait arbitraire. La structure modulaire et la représentation graphique de la plate-forme se devait de faciliter grandement la supervision mais aussi la définition et la révision d'une machine donnée. Les robots joueurs de soccer ont permis de le vérifier puisque les divers comportements présents dans cette machine ont continuellement été revus et améliorés sans nécessiter le moindre changement des étages décisionnels au-dessus. De la même façon, les étages décisionnels ont pu être revus, la structure de la machine reconfigurée, sans nécessiter de changement au sein des comportements. Les caractéristiques graphiques de l'architecture ne sont toutefois pas exploitées à leur plein potentiel pour l'instant. Finalement l'architecture a été programmée à un niveau logiciel générique de façon à permettre le développement multi-plateforme. L'utilisation du C++ avec une programmation par objets a permis d'obtenir ce niveau générique et de le tester entre autre sur le système d'exploitation *Windows* en simulation et sur *Linux* sur les robots joueurs de soccer.

Selon les objectifs et les spécifications établies préalablement, il apparaît donc que la machine décisionnelle hiérarchique avec mécanisme de supervision décisionnelle réponde dans l'ensemble très bien aux éléments requis. D'ailleurs, si l'on se fie aux critères d'évaluation couramment utilisés dans la littérature, la machine décisionnelle développée semble se tirer d'affaire assez bien. Par exemple, R.C. Arkin ([6]) identifie certains critères dont les suivants, spécifiques aux architectures dites « behavior-based », concept semblable à celui développé :

- Support pour le parallélisme : pour cet aspect la MDH diffère par son utilisation d'une ligne décisionnelle unique.
- Capacité à cibler du matériel : la machine se veut indépendante du matériel et devrait être en mesure de s'implanter sur tout type de matériel (niveau logiciel générique).
- Support pour la modularité : un des critères essentiels de la MDH, elle est modulaire et le bénéfice de ce critère a été démontré.
- Robustesse : la machine comme tel, ainsi que la supervision décisionnelle, assurent un fonctionnement robuste au niveau du module de cognition d'un robot.

- Flexibilité en cours d'exécution : le mécanisme décisionnel, capable de changer de ligne décisionnelle instantanément, est très dynamique et flexible. De plus, des possibilités d'adaptation et d'apprentissage, en temps réel, sont facilement possibles avec l'architecture proposée.
- Efficacité au niveau de la performance : les mesures de performance n'ont démontré aucun signe de problème à ce niveau.

Alami et coll. ([4]) identifient six propriétés qui se devraient d'être intrinsèques à toute architecture hybride : programmabilité, autonomie et adaptabilité, réactivité, consistance comportementale, robustesse et extensibilité. Ces six propriétés sont effectivement des éléments présents dans la machine décisionnelle hiérarchique, avec un bémol pour l'adaptabilité qui n'a été que très peu étudiée.

5.5 Améliorations possibles

En cours de développement, plusieurs nouvelles fonctionnalités et améliorations possibles ont été notées afin de permettre au concept d'aller plus loin. Voici les principaux points notés :

- Sous-utilisation des capacités graphiques : grâce à la représentation graphique possible d'une machine décisionnelle, plusieurs utilitaires et outils de développements pourraient se greffer éventuellement aux éléments développés. Par exemple, pour la définition et la modification d'une machine décisionnelle, une interface graphique serait facilement développable. Ce même genre d'interface pourrait être utilisé lors de l'implantation de supervision décisionnelle avec superviseur humain. Il y aurait également d'autre possibilité et pour l'instant donc ces capacités sont sous-utilisées.
- Structure de la machine décisionnelle fixe en cours d'exécution : une fois une machine décisionnelle développée, sa structure se doit de demeurer fixe en cours d'exécution. Aucun nœud ou action ne peuvent être greffés, supprimés ou reconnectés au sein de la machine. Cette possibilité pourrait se développer relativement facilement et être grandement utile pour les développements de mécanismes cognitifs plus complexes que ceux développés sur le système expérimental présenté (adaptatifs, apprentissage).
- Manque d'outils génériques au sein de l'implantation logicielle : de nombreux outils pourraient se retrouver assez fréquemment dans diverses machines spécifiques,

indépendamment de l'application visée. Par exemple, des mécanismes de communication standardisés, des méthodes d'apprentissage et etc. Ces outils peuvent toujours être greffés par le développeur d'une machine spécifique, mais s'il est possible d'offrir des solutions suffisamment génériques, elles devraient être incluses au sein de l'implantation logicielle proposée pour la machine décisionnelle hiérarchique.

- Utilisation de mémoire exagérée : il a été observé dans les mesures que l'utilisation de la mémoire vive peut croître considérablement lorsque le nombre de nœuds présents dans une machine augmente. Dans l'objectif de permettre une mise à l'échelle de l'implantation logicielle de la MDH, par exemple sur un système à microcontrôleur offrant une quantité minimale de ressources, ce problème devrait être traité.

Voici donc quelques points plus faibles de la machine décisionnelle développée. Ces points devraient être considérés lors de développements futurs et lors de mises à jour de l'implantation logicielle proposée.

CONCLUSION

Les travaux documentés dans le présent mémoire se sont échelonnés sur de nombreuses semaines de rédaction mais surtout sur de nombreux mois de développements, d'essais et de compétitions. Faire fonctionner, en temps réel et de façon autonome, une équipe de robots présentant un comportement prévisible, fiable et robuste n'est clairement pas une simple tâche. Ainsi, il a été nécessaire de travailler avec acharnement au développement et à l'intégration de divers modules et composantes permettant de compléter les fonctionnalités désirées du système multi-robots expérimental. De plus, la présentation d'une société technique en compétition internationale du calibre de la *RoboCup* demande, en plus des aptitudes techniques, une motivation exemplaire afin d'assurer le financement et la gestion des divers aspects administratifs de cette société.

Plus concrètement, une architecture décisionnelle hiérarchique adressant la problématique de la prise de décision en temps réel distribuée au sein d'un système multi-robots coopératif a été développée. L'architecture développée, la machine décisionnelle hiérarchique, est une architecture comportant plusieurs caractéristiques intéressantes : capacités réactives et délibératives, modularité, robustesse, représentation graphique simple mais puissante, nombre de nœuds et de hiérarchies arbitraires, niveau logiciel générique. Afin d'adresser la problématique de la coopération comme tel, il est possible d'utiliser le concept de règles préétablies. De plus, de façon à régler des conflits décisionnels qui sont sur certains systèmes inévitables, le concept de supervision décisionnelle a été introduit. Grâce à une communication de type clients/serveurs, il est donc à ce moment possible d'intervenir au niveau de la prise de décision distribuée, de la superviser, et ce sur toute la hiérarchie décisionnelle d'une machine donnée.

Des robots joueurs de soccer ont donc servi de banc d'essai et l'ensemble des fonctionnalités du système multi-robots développé ont été validées par deux compétitions dans la « Middle Size Robot League » de la *RoboCup*. La *RoboCup* 2005 a démontré que le projet *Robofoot ÉPM* détenait une expertise suffisante dans tous les domaines reliés au soccer robotisé. Le système présenté en compétition offrait effectivement toutes les fonctionnalités des meilleures équipes. La plateforme à vitesses différentielles désuète pour le développement de capacités offensives ainsi que le manque de raffinement des comportements offensifs ont cependant nuit aux performances

compétitives de l'équipe. Pour ce qui est des challenges techniques, *Robofoot ÉPM* s'est très bien illustrée pour une nouvelle équipe et elle devrait s'assurer de continuer, pour les compétitions futures, à démontrer son sérieux scientifique en offrant des prestations respectables lors de ces challenges. Pour ces diverses compétitions, la machine décisionnelle développée a été d'une très grande utilité. La robustesse, la modularité et la versatilité de la machine développée ont grandement facilité le travail de l'équipe afin de s'ajuster au niveau de jeu élevé de la ligue ainsi qu'aux diverses règles régissant les tournois.

Ce projet visait initialement à établir les bases pour de développements futurs et conséquemment plusieurs aspects plus poussés, en termes d'intelligence artificielle par exemple, n'ont pas encore été traités. Ainsi, pour espérer bien situer l'architecture décisionnelle développée au sein du riche domaine des systèmes multi-robots coopératifs, il serait nécessaire dans l'avenir d'étudier le fonctionnement de la machine décisionnelle hiérarchique dans diverses implantations aux caractéristiques variées.

6.2 Travaux futurs

En plus des améliorations possibles présentées au chapitre précédent, voici une liste d'éléments qui devraient faire l'objet de développements futurs et ce idéalement dès les mois et non les années à venir. Ces travaux auraient pour but de greffer à la machine décisionnelle hiérarchique de nouvelles fonctionnalités, d'améliorer celles déjà implantées et en quelque sorte de continuer de bâtir, d'ajouter d'autres concepts et modules grâce aux bases structurales établies dans le présent projet.

6.2.1 Caractéristiques graphiques

Les caractéristiques graphiques de la MDH sont intéressantes et jusqu'à présent sous-utilisées. Une architecture comme *XABSL* (Lötzsch et coll., [25]) constitue un excellent exemple d'utilisation assez poussée des caractéristiques graphiques. Jumelée avec l'utilisation du XML comme langage générique pour la définition de machines à états, cette architecture propose des outils de développement très puissants. Certains outils logiciels peuvent être très intéressants pour la génération automatique de documentation. Par exemple, *Doxygen* permet une documentation automatique du code source, schémas à l'appui. De plus, l'utilitaire *Dot* de *Graphviz* permettrait facilement de représenter automatiquement une machine décisionnelle développée. Pour illustrer

cette possibilité, la machine *Soccerteam_HDM* ainsi que les patrons *PrepareTeam* et *SoccerPlayer* ont été représentés grâce à ces outils. Les figures obtenues sont présentées en Annexe 1. Les caractéristiques graphiques de la MDH devraient permettre de développer les éléments suivants :

- Automatisation de la documentation des MDHs
- Interface graphique de modélisation de machines décisionnelles hiérarchiques
- Interface graphique de supervision décisionnelle humaine

6.2.2 Supervision décisionnelle approfondie

La supervision décisionnelle développée sur les robots joueurs de soccer a démontré sa faisabilité et son utilité. Les définitions de vecteur décisionnel, matrice décisionnelle, vecteur de supervision et matrice de supervision sont très utiles à cet effet. Cependant, certaines particularités du processus de supervision n'ont été que peu ou pas étudiées. Par exemple, avec des robots à prise de décision distribuée et communication asynchrone avec le superviseur, les aspects de détection de conflits et de propagation de supervision sont particuliers.

La supervision décisionnelle est donc encore dans un état embryonnaire. Son fonctionnement et son implantation en fonction de différentes architectures de communication devront être approfondis. De plus, le mécanisme de supervision a été prévu au sein des nœuds lors de l'implantation logicielle de la machine décisionnelle hiérarchique, mais encore aucune implantation de superviseur se voulant générique n'a été greffée à cette implantation logicielle. Seul un exemple avec les robots joueurs de soccer a été développé.

6.2.3 Développement de machines décisionnelles pour applications diverses

La machine décisionnelle n'a pour l'instant été testée que sur un seul système expérimental, les robots joueurs de soccer. Quoique ses possibilités semblent intéressantes, il sera nécessaire de le confirmer et d'en parfaire ses fonctionnalités en la testant sur diverses implantations de systèmes multi-robots, quelle qu'en soit l'application visée. On sait que les systèmes multi-robots pourront éventuellement jouer un rôle intéressant dans divers domaines (aérospatiale, agriculture, industrie manufacturière, etc.) et toute application spécifique présentera des caractéristiques d'implantation originales. La machine décisionnelle développée devrait donc être en mesure d'assurer un

fonctionnement adéquat pour un grand nombre, voire la totalité si possible, des diverses applications visées.

En particulier, la machine décisionnelle devrait être en mesure de répondre à des besoins très variés au niveau des éléments suivants :

- Mécanismes cognitifs allant des plus simples aux plus complexes
- Implantation de machines décisionnelles sur des architectures informatiques variées
- Matériel sensoriel et actuateurs liés au module de cognition très variés
- Nombre de robots des systèmes développés variable et pouvant être relativement grand
- Architectures de communication variées (configurations clients/serveurs arbitraires)

6.2.4 Nouvelle plateforme de robot joueur de soccer

Le projet *Robofoot* compte développer dans les prochains mois une nouvelle plateforme ayant pour but d'éliminer la contrainte non-holonome de la plateforme actuelle. Ce type de plateforme, dite omnidirectionnelle, est déjà utilisé par quelques unes des meilleures équipes de la « Middle Size Robot League ». Au niveau du contrôle de ballon et donc des capacités offensives, une plateforme omnidirectionnelle procure un avantage dominant face à la plateforme à vitesses différentielles.

Le développement de plateformes mécatroniques n'a aucun lien direct avec le développement d'architectures décisionnelles pour systèmes multi-robots. Cependant, la machine décisionnelle développée pour le projet *Robofoot* a prouvé son efficacité et sera conservée avec la nouvelle plateforme. Cette nouvelle plateforme devient donc un test intéressant permettant de démontrer la modularité et l'aspect générique de la machine décisionnelle hiérarchique, ainsi que son aptitude à gérer un système multi-robots hétérogène. En effet, il est prévu d'intégrer dans un premier temps un seul exemplaire de robot omnidirectionnel au sein de l'équipe de robots déjà développée et présentée en compétitions. Ainsi, au niveau de la machine décisionnelle déjà développée, seules les actions ainsi que certains comportements devraient demander des modifications. Par exemple, les paramètres d'action pourraient être différents et les comportements rattachés au contrôle du ballon devront être revus. Ceci dit, tout le reste de la machine décisionnelle devrait pouvoir être réutilisé sans aucune modification. Voilà qui permettrait de sauver énormément de temps de développement et qui démontrerait l'aspect générique souhaité.

BIBLIOGRAPHIE

[1] Advanced Cruise-Assist Highway System Research Association. *AHSRA Web Site*. Advanced Cruise-Assist Highway System Research Association. [En ligne, octobre 2005. http://www.ahsra.or.jp/index_e.html]

[2] Agence Spatiale Canadienne. *Apogee – March 2002 : Canada is Going to Mars*. Agence Spatiale Canadienne, 2002.

[3] R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert. *Multi Robot Cooperation in the Martha Project*. Dans *IEEE Robotics and Automation Magazine*, Vol. 5, No. 1. IEEE. 1997, pp.36-45.

[4] R. Alami, R. Chatila, S. Fleury, M.Ghallab, F. Ingrand. *An Architecture for Autonomy*. Dans *International Journal of Robotics Research*, Vol. 17, No. 4. Sage Publications, 1998, pp.315-337.

[5] E.C. Aldridge Jr., C.S. Fiorina, M.P. Jackson, L.A. Leshin, L.L. Lyles, P.D. Spudis, N.D. Tyson, R.S. Walker, M.T. Zuber. *A Journey to Inspire, Innovate and Discover*. Commission on Implementation of United States Space Exploration Policy, 2004.

[6] R.C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, 1998.

[7] J. Beaudry. *Projet SpinoS : conception et contrôle d'un robot mobile à vitesses différentielles*. Projet de fin d'études, École Polytechnique de Montréal, 2001.

[8] M. Bowling, M. Veloso. *Motion Control in Dynamic Multi-Robot Environments*. Proceedings of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation, 1999, pp.220-230.

[9] R.A. Brooks. *A Robust Layered Control System for a Mobile Robot*. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1985.

[10] Carnegie Mellon University. *Life in the Atacama*. Carnegie Mellon University - Robotics Institute, NASA - ASTEP. [En ligne, octobre 2005. http://www.frc.ri.cmu.edu/atacama/]

[11] S. Carpin, L.E . Parker. *Cooperative Leader Following in a Distributed Multi-Robot System*. Proceedings of the 2002 IEEE International Conference on Robotics & Automation, 2002, pp.2994-3001.

[12] L. Chainowicz, M.F.M. Campos, V. Kumar. *Dynamic Role Assignment for Cooperative Robots*. Proceedings of the 2002 IEEE International Conference on Robotics & Automation, 2002, pp.292-298.

[13] L. Chainowicz, T. Sugar, V. Kumar, M.F.M. Campos. *An Architecture for Tightly Coupled Multi-Robot Cooperation*. Proceedings of the 2001 IEEE International Conference on Robotics & Automation. 2001, pp.2292-2297.

[14] R. M. DeSantis. *Dynamique des systèmes mécaniques sous contraintes holonomes et non holonomes*. École Polytechnique de Montréal. 1999.

[15] A. Egorova. *MAAT - Multi Agent Authoring Tool for Programming Autonomous Mobile Robots*. Thèse M.Sc.A., Institut für Informatik, Freie Universität Berlin, 2004.

[16] R. Emery, K. Sikorsky, T. Balch. *Protocols for Collaboration, Coordination and Dynamic Role Assignment in a Robot Team*. Proceedings of the 2002 IEEE International Conference on Robotics & Automation, 2002, pp.3008-3015.

[17] B. O. Gallmeister. *POSIX 4 : Programming for the Real World*. O'Reilly and Associates. 1995.

[18] E. Gat. *On Three-Layer Architectures*. Dans D. Kortenkamp et coll. *AI and Mobile Robots*. AAAI Press, 1998, pp.195-210.

[19] D. van Heesch. *Doxygen*. D. van Heesch. [En ligne, octobre 2005. http://www.stack.nl/~dimitri/doxygen/]

[20] T.L. Huntsberger, P. Pirjanian , A. Trebi-Ollennu, H. Das, H. Aghazarian, A.J. Ganino, M.S. Garrett, S.S. Joshi, and P.S. Schenker. *CAMPOUT: A Control Architecture for Tightly Coupled Coordination for Multi-robot Systems for Planetary Surface Exploration*. IEEE Transactions on Systems, Man, and Cybernetics, Volume 33, Number 5, 2003, pp.555-559.

[21] Institut de recherche d'Hydro-Québec. *Modules Intégrés de Contrôle de ROBots*. Institut de recherche d'Hydro-Québec. [En ligne, octobre 2005. http://www.robotique.ireq.ca/new/]

[22] R. L'Archevêque, E. Dupuis. *Autonomous Robotics and Ground Operations*. Proceedings of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space – iSAIRAS 2003. 2003.

[23] M. Lemay, F. Michaud, D. Létourneau, J.-M. Valin. *Autonomous initialization of robot formations*. Proceedings of the 2004 IEEE International Conference on Robotics & Automation, 2004, pp.3018-3023.

[24] M. Lötzsch. *The Extensible Agent Behavior Specification Language.* Artificial Intelligence Research Group, Humboldt-University of Berlin. [En ligne, octobre 2005. http://www.ki.informatik.hu-berlin.de/XABSL/]

[25] M. Lötzsch, J. Bach, H.-D. Burkhard, M. Jüngel. *Designing Agent Behavior with the Extensible Agent Behavior Specification Language XABSL.* Dans *RoboCup 2003: Robot Soccer World Cup VII).* Lecture Notes in Artificial Intelligence. Springer, 2004, à paraître.

[26] MSL Technical Committee. *Middle Size Robot League Rules and Regulations for 2005.* The RoboCup Federation. [En ligne, octobre 2005. http://www.er.ams.eng.osaka-u.ac.jp/robocup-mid/index.cgi?page=Rules+and+Regulations]

[27] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin,_Won Soo Kim. *CLARAty: An Architecture for Reusable Robotic Software.* SPIE Aerosense Conference, 2003, pp.121-132.

[28] N. Noguchi, J. Will, J. Reid. Q. Zhang. *Development of a master-slave robot system for farm operations.* Dans *Computers and electronics in agriculture,* Vol. 44. Elsevier, 2004, pp.1-19.

[29] Office québécois de la langue française. Le *grand dictionnaire terminologique.* Office québécois de la langue française. [En ligne, octobre 2005. http://www.oqlf.gouv.qc.ca/ressources/gdt.html]

[30] F.C. Olson. *Selecting Landmarks for Localization in Natural Terrain.* Dans *Autonomous Robots 12.* Kluwer Academic Publishers, 2002, pp.201-210.

[31] L. E. Parker, *ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation,* IEEE Transactions on Robotics and Automation, 1998, pp.220-240.

[32] T. Pilarski, M. Happold, H. Pangels, M. Ollis, K. Fitzpatrick, and A. Stentz. *The DEMETER System for Autonomous Harvesting.* Autonomous Robots, Vol. 13, No. 1, 2002, pp.9-20.

[33] P. Pirjanian, T.L. Huntsberger, A. Trebi-Ollennu, H. Aghazarian, H. Das, S. Joshi, and P.S. Schenker. *CAMPOUT: A Control Architecture for Multi-robot Planetary Outposts.* Proceedings of the SPIE Symposium on Sensor Fusion and Decentralized Control in Robotic Systems III, Vol. 4196, 2000, pp.221-230.

[34] The RoboCup Federation. *RoboCup Official Site.* The RoboCup Federation. [En ligne, octobre 2005. http://www.robocup.org]

[35] RoboCup Project FU Berlin. *FU-Fighters RoboCup Team*. AG Künstliche Intelligenz, Institut für Informatik, FU Berlin. [En ligne, octobre 2005. http://robocup.mi.fu-berlin.de/]

[36] Robofoot ÉPM. *Site web du projet Robofoot*. Robofoot ÉPM. [En ligne, octobre 2005. http://robofoot.polymtl.ca]

[37] R. Siegwart, I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2004.

[38] P. Song, V. Kumar. *A Potential Field Based Approach to Multi-Robot Manipulation*. Proceedings of the 2002 IEEE International Conference on Robotics & Automation, 2002, pp.1217-1222.

[39] P. Stone. *Layered Learning in Multiagent Systems*. The MIT Press, 2000.

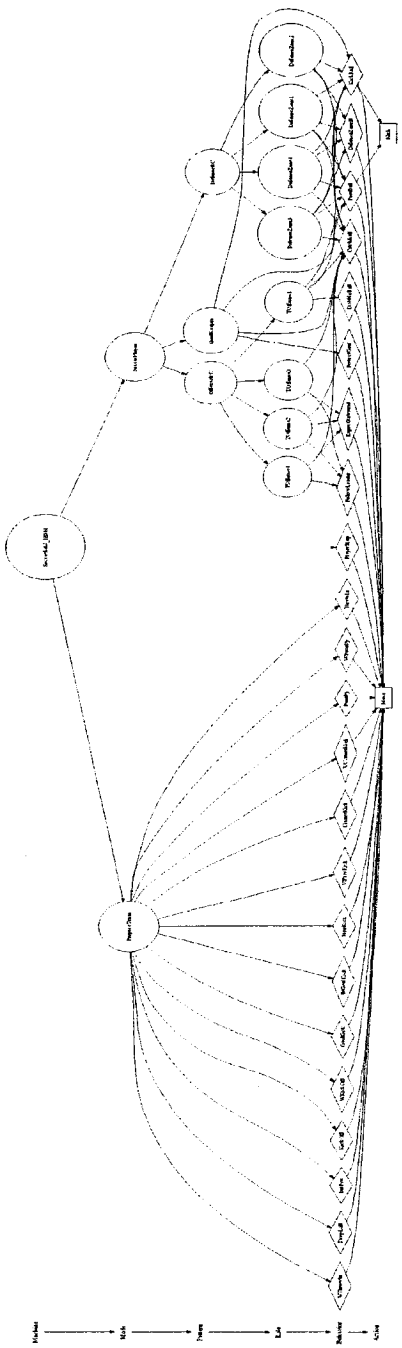
[40] Y. Takahashi, K. Edazawa, and M. Asada. *Multi-Module Learning System for Behavior Acquisition in Multi-Agent Environment*. Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002, pp.435-442.

[41] A. Trebi-Ollennu. *Robot Work Crew Web Page*. NASA Jet Propulsion Laboratory. [En ligne, octobre 2005. http://prl.jpl.nasa.gov/projects/rwc/rwc_index.html]

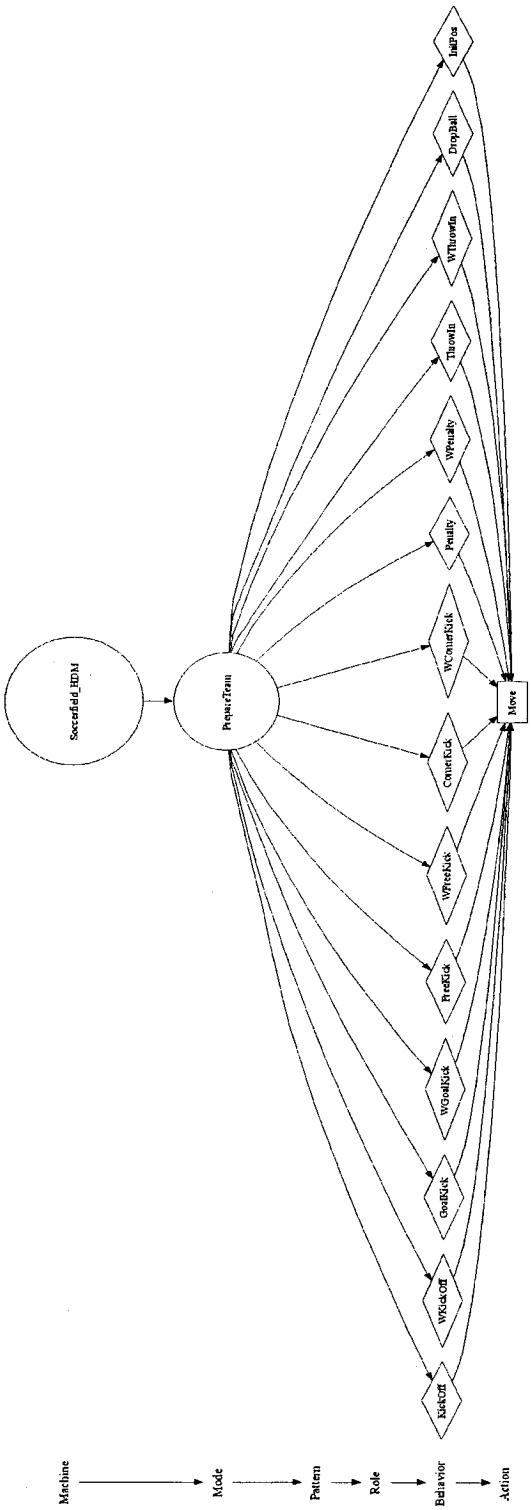
[42] Y. Uny Cao, A. S. Fukunaga, A. B. Kahng. *Cooperative Mobile Robotics: Antecedents and Directions*. UCLA Computer Science Department, 1995.

ANNEXE A - UTILISATION DES OUTILS *DOT* ET *GRAPHVIZ*

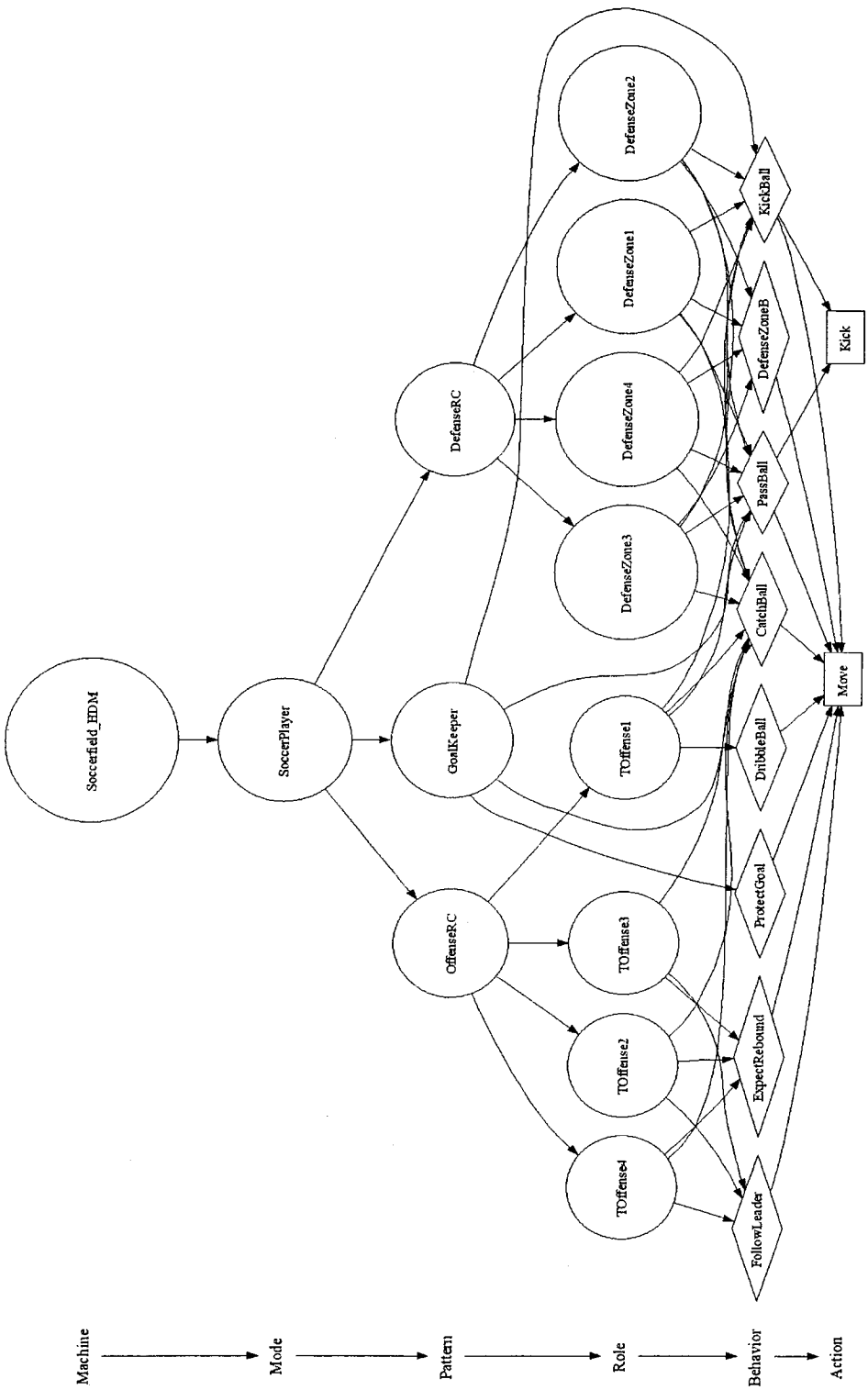
Voici des schémas de machines décisionnelles générés automatiquement grâce à ces outils.



a) Machine *Soccerteam_HDM*



b) Mode *PrepareTeam*



c) Mode *SoccerPlayer*

ANNEXE B - CODE SOURCE DE LA MACHINE DÉCISIONNELLE HIÉRARCHIQUE

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile robots with distributed sensing,cognition and control. Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: HDM_node.h,v 1.3 2005/06/05 22:24:19 julien Exp $
 *
 */

#ifndef HDM_NODE_H
#define HDM_NODE_H

//! \mainpage Hierarchical Decision Machine \n for Multi-robot Systems
//! The Hierarchical Decision Machine project is part of the Robofoot's project
which develops a multi-robot system able to play soccer autonomously. This team
of robots needs to take decisions in real-time, with distributed perception and
cognition. These decisions must consider the teammates and opponents. In order
to coordinate their actions and show cooperative behaviors, the multi-robot
system needs appropriate decision mechanisms. Since the cognition and
perception are distributed, coordination problems can frequently occur if no
special mechanisms are used. Here, we use the supervised decision concept which
```

```
allows a supervisor to dictate the decision for a specific robot and on the
whole hierarchy of decisions.
//!  \n
//! \n
//!  Robofoot's website: http://robofoot.polymtl.ca

//! \class HDM_node
//! \brief Contains the interface of the \c HDM_node virtual class.

#include "HDM.h"
#include "Creator.h"

enum NODE_TYPE {DECISION, BEHAVIOR, TREE};

class HDM_node
{
private:
    void get_pointers();
    int pointers_ok;

protected:
    char *name;
    int id;
    int node_type;

    Creator * creator;
    BrainController * braincontroller;
    cPerception * perception;
    DataAcquisition * data;

    struct Robot_info rbt_info;
    struct Field_info field_info;
    double period;
    Configuration config;

public:
    HDM_node();

    ~HDM_node();

    virtual void behavior() = 0;

    virtual HDM_node *decision(int ID=-1) = 0;

    virtual void update_info();

    void set_name(char *name_) {name = name_;};

    void set_id(int id_) {id = id_;};

    char *get_name(){return name;};

    int get_id(){return id;};

    int get_type(){return node_type;};

    virtual void set_field_info(const struct Field_info fi){field_info = fi;};

    virtual void set_robot_info(const struct Robot_info rbt);
```

```
};
#endif
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
// mobile robots with distributed sensing,cognition and control. Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: HDM_node.C,v 1.5 2005/06/11 20:28:40 julien Exp $
 *
 */

#include "HDM_node.h"


HDM_node::HDM_node()
{
    creator = NULL;
    perception = NULL;
    data = NULL;
    creator = Creator::Instance();
    pointers_ok = 0;

    id = 0;

    char tmpStr[1024], *repertoire;

    repertoire = getenv("REP_CONFIG");
    if (repertoire !=NULL)
    {
        strcpy(tmpStr, repertoire);
```

```
        strcat(tmpStr, "player.cnf");
    }
    else
        strcpy(tmpStr, "player.cnf");

    // Lire le fichier de configuration pour obtenir la periode
    if (config.read(tmpStr)==0)
    {
        printf("FATAL ERROR : Unable to open file <%s>!\n", tmpStr);
    }

    period = config.get_matrix("period", Configuration::MIC_ERROR);
    get_pointers();
}

HDM_node::~HDM_node(){}

void HDM_node::get_pointers()
{
    pointers_ok = 1;
    braincontroller = creator->GetBrainController();
    perception = creator->GetPerception();
    data = creator->GetDataAcquisition();
    if(braincontroller == NULL)
    {
        cerr<<"Unable to obtain BrainController instance in HDM_node"<<endl;
        pointers_ok = 0;
    }
    if(perception == NULL)
    {
        cerr<<"Unable to obtain Perception instance in HDM_node"<<endl;
        pointers_ok = 0;
    }
    if(data == NULL)
    {
        cerr<<"Unable to obtain DataAcquisition instance in HDM_node"<<endl;
        pointers_ok = 0;
    }
}

void HDM_node::update_info()
{
    if(!pointers_ok) get_pointers();
    else
    {
        cPosition MyPosition;
        cVitesse MySpeed;
        cPosition BallPosition;
        perception->GetPosition(MyPosition);
        perception->GetVitesse(MySpeed);
        perception->GetPositionObjet("BALLONORANGE", BallPosition);

        rbt_info.rbt_pos.x = MyPosition.X;
        rbt_info.rbt_pos.y = MyPosition.Y;
        rbt_info.rbt_pos.theta = MyPosition.Orientation;
        int id = braincontroller->GetRobotID();
        rbt_info.rbt_pos.id = id;
        if(id >= NB_PLAYERS/2) rbt_info.team = 1;
        else rbt_info.team = 0;
```

```
        int team = rbt_info.team;
        rbt_info.vtan = MySpeed.Vtan;
        rbt_info.omega = MySpeed.Omega;
        rbt_info.ball_pos.x = BallPosition.X;
        rbt_info.ball_pos.y = BallPosition.Y;
        rbt_info.ball_pos.theta = BallPosition.Orientation;
        rbt_info.ball_v = 0;
        rbt_info.ball_dir = rbt_info.ball_pos.theta;

        // TODO: obtenir positions des autres robots de perception
        int i;
        int crt = 0;
        for(i=0; i<(NB_PLAYERS/2); i++)
        {
            if(id!=i+4*team)
            {
                rbt_info.teammates[crt].x = OUTSIDE;
                rbt_info.teammates[crt].y = OUTSIDE;
                rbt_info.teammates[crt].theta = OUTSIDE;
                rbt_info.teammates[crt].id = i+4*team;
                crt++;
            }
        }
        for(i=0; i<NB_PLAYERS/2; i++)
        {
            rbt_info.opponents[i].x = OUTSIDE;
            rbt_info.opponents[i].y = OUTSIDE;
            rbt_info.opponents[i].theta = OUTSIDE;
            rbt_info.opponents[i].id = i+4-4*team;
        }
    }
}

void HDM_node::set_robot_info(const struct Robot_info rbt)
{
    rbt_info = rbt;

    // on ne veut pas jouer avec des ballons en dehors en Y!
    // alors pkoi pas simplement ne pas le dire au robot!
    if(rbt_info.ball_pos.y > field_info.field_l/2)
        rbt_info.ball_pos.y = field_info.field_l/2-0.1;
    if(rbt_info.ball_pos.y < -field_info.field_l/2)
        rbt_info.ball_pos.y = -field_info.field_l/2+0.1;
};
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile robots with distributed sensing,cognition and control. Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: HDM_tree.h,v 1.3 2005/05/28 18:52:19 pmf Exp $
 *
 */

#ifndef HDM_TREE_H
#define HDM_TREE_H

//! \class HDM_tree
//! \brief Contains the interface of the \c HDM_tree abstract class.

#include <malloc.h>
#include "HDM.h"
#include "HDM_node.h"
#include "Action.h"

class HDM_tree : public HDM_node
{
private:

protected:
    int nb_layers;
    Nlist layer_names;
    Nlist node_list;
```

```
    Nlist action_list;
    HDM_node * maindecision;
    HDM_node * currentnode;
    int *supervised_decision;
    int *current_decision;
/*
    void get_pointers();
    int pointers_ok;
    Creator * creator;
    cPerception * perception;
    DataAcquisition * data;
*/    int data_output;

public:
    HDM_tree();
    ~HDM_tree();

    virtual int build() = 0;

    virtual Nlist process();

    void behavior() {};

    HDM_node *decision(int ID=-1) {return NULL;};

    void set_maindecision(HDM_node *md) {maindecision = md;};

    void set_maindecision(const int ID) {reset_decision(); maindecision =
(HDM_node *)node_list.GetData(ID);};

    void set_decision(int *d) {supervised_decision = d;};

    int * get_decision() {return current_decision;};

    void reset_decision();

    virtual void set_field_info(const struct Field_info fi);

    virtual void set_data_output(const int data_output_){data_output =
data_output_;};

    virtual int get_nb_layers(){return nb_layers;};
};
#endif
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile robots with distributed sensing,cognition and control. Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: HDM_tree.C,v 1.3 2005/05/28 18:52:19 pmf Exp $
 *
 */

#include "HDM_tree.h"

HDM_tree::HDM_tree()
{
    data_output = 0;
    node_type = TREE;
}

HDM_tree::~HDM_tree()
{
    while(!layer_names.IsEmpty())
    {
        delete (string *)layer_names.Remove();
    }
    while(!node_list.IsEmpty())
    {
        delete (HDM_node *)node_list.Remove();
    }
    while(!action_list.IsEmpty())
```

```
    {
        delete (Action *)action_list.Remove();
    }
    delete supervised_decision;
    delete current_decision;
}

Nlist HDM_tree::process()
{
    layer_names.Front();
//     cerr<<"HDM main decision, "<<layer_names.GetName()<<" : "<<maindecision-
>get_name()<<endl;

    int i;

    // pour verifier jusqu'a quel niveau la decision se rend (jusqu'au
behavior)
//    cerr<<"SD vector: ";
    for(i=0;i<nb_layers;i++)
    {
        current_decision[i] = -1;
//        cerr<<supervised_decision[i]<<" ";
    }
//    cerr<<endl;

    // supervised decision on first layer
    if(supervised_decision[0] != -1)
        maindecision = (HDM_node *)node_list.GetData(supervised_decision[0]);
    currentnode = maindecision;
    current_decision[0] = currentnode->get_id();

    int done = 0;
    i = 1;
//    for(int i=1; i<nb_layers-1; i++)
    while(currentnode->get_type() != BEHAVIOR)
    {
        currentnode->set_robot_info(rbt_info);
        currentnode = currentnode->decision(supervised_decision[i]);
        current_decision[i] = currentnode->get_id();

        layer_names.Next();
/*
        if(supervised_decision[i] != -1)
        {
            cerr<<"Supervised           decision           on           layer:
"<<layer_names.GetName()<<endl;
            cerr<<"\tID:  "<<supervised_decision[i]<<"\tNode:  "<<  currentnode-
>get_name()<<endl;;
        }
*/
        //! \todo TODO: revoir ce mecanisme en utilisant DataAcquisition
        if(data_output)
        {
//                data->add(layer_names.GetName(), currentnode->get_id());
        }

//            cerr<<"HDM layer "<<i<<" decision, "<<layer_names.GetName()<<" :
"<<currentnode->get_name()<<endl;
        i++;
```

```
    }
    currentnode->set_robot_info(rbt_info);
    currentnode->behavior();


    // TODO: verifier si on ressoirt ou non la decision actuelle
    Nlist tmp;
    return tmp;
};

void HDM_tree::set_field_info(struct Field_info fi)
{
    field_info = fi;

//    maindecision->set_field_info(fi);

    node_list.Front();
    int i;
    for(i = 0; i<node_list.GetNbElements(); i++)
    {
        ((HDM_node *)node_list.GetData())->set_field_info(fi);
        node_list.Next();
    }

    action_list.Front();
    for(i = 0; i<action_list.GetNbElements(); i++)
    {
        ((Action *)action_list.GetData())->set_field_info(fi);
        action_list.Next();
    }

}

void HDM_tree::reset_decision()
{
    for(int i=0; i<nb_layers ; i++)
        supervised_decision[i] = -1;
}
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile  robots  with  distributed  sensing,cognition  and  control.  Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: Decision_node.h,v 1.2 2005/05/28 18:52:19 pmf Exp $
 *
 */

#ifndef DECISION_NODE_H
#define DECISION_NODE_H

//! \class Decision_node
//! \brief Contains the interface of the \c Decision_node virtual class.

#include "HDM_node.h"

class Decision_node: public HDM_node
{
protected:
    Nlist node_list;
    HDM_node *current;
    HDM_node *previous;

public:
    Decision_node();
    ~Decision_node();
```

```
    virtual HDM_node *decision(int ID=-1) = 0;

    void behavior() {};

    virtual void set_field_info(const struct Field_info fi);
};
#endif
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile robots with distributed sensing,cognition and control. Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: Decision_node.C,v 1.2 2005/05/28 18:52:19 pmf Exp $
 *
 */

#include "Decision_node.h"


Decision_node::Decision_node() { node_type = DECISION;}

Decision_node::~Decision_node()
{
    while(!node_list.IsEmpty())
    {
        delete (HDM_node *)node_list.Remove();
    }
}

void Decision_node::set_field_info(struct Field_info fi)
{
    field_info = fi;

    node_list.Front();
    for(int i = 0; i<node_list.GetNbElements(); i++)
```

```
    {
        ((HDM_node *)node_list.GetData())->set_field_info(fi);
        node_list.Next();
    }
}
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile  robots  with  distributed  sensing,cognition  and  control.  Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: Behavior_node.h,v 1.2 2005/05/28 18:52:19 pmf Exp $
 *
 */

#ifndef BEHAVIOR_NODE_H
#define BEHAVIOR_NODE_H

//! \class Behavior_node
//! \brief Contains the interface of the \c Behavior_node virtual class.

#include "HDM_node.h"
#include "Action.h"

class Behavior_node: public HDM_node
{
protected:
    Nlist action_list;
    struct Action_parameters action_parameters;

public:
    Behavior_node();
    ~Behavior_node();
```

```
    void behavior();

    virtual void update_params()=0;

    void execute_actions();

    HDM_node *decision(int ID=-1) {return NULL;};

    virtual void set_field_info(const struct Field_info fi);
};
#endif
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile robots with distributed sensing,cognition and control. Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: Behavior_node.C,v 1.2 2005/05/28 18:52:19 pmf Exp $
 *
 */

#include "Behavior_node.h"


Behavior_node::Behavior_node() { node_type = BEHAVIOR; };

Behavior_node::~Behavior_node()
{
    while(!action_list.IsEmpty())
    {
        delete (Action *)action_list.Remove();
    }
}

void Behavior_node::behavior()
{
//    HDM_node::update_info();
    update_params();
    execute_actions();
}
```

```
void Behavior_node::execute_actions()
{
    action_list.Front();

    for(int i = 0; i<action_list.GetNbElements(); i++)
    {
        ((Action *)action_list.GetData())->set_robot_info(rbt_info);
        ((Action *)action_list.GetData())->action(action_parameters);
        action_list.Next();
    }
}

void Behavior_node::set_field_info(struct Field_info fi)
{
    field_info = fi;

    action_list.Front();
    for(int i = 0; i<action_list.GetNbElements(); i++)
    {
        ((Action *)action_list.GetData())->set_field_info(fi);
        action_list.Next();
    }
}
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile  robots  with  distributed  sensing,cognition  and  control.  Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: Action.h,v 1.2 2005/05/28 18:52:19 pmf Exp $
 *
 */

#ifndef ACTION_H
#define ACTION_H

//! \class Action
//! \brief Contains the interface of the \c Action base class.

#include "HDM.h"
#include "Creator.h"

class Action
{
private:
    void get_pointers();
    int pointers_ok;

protected:
    Creator * creator;
    BrainController * braincontroller;
    cPerception * perception;
```

```
    Controller * ctrl;
    MotionControl * motion;
    DataAcquisition * data;
    struct Robot_info rbt_info;
    struct Field_info field_info;
    Configuration config;

public:
    Action();
    virtual ~Action(){};

    virtual void action(Action_parameters parameters);

    virtual void update_info();

    virtual void set_field_info(const struct Field_info fi){field_info = fi;};

    virtual void set_robot_info(const struct Robot_info rbt){rbt_info = rbt;};
};
#endif
```

```
/* -*- Mode: C++; c-file-style: "stroustrup"; indent-tabs-mode: nil -*- */

// This file is part of the Hierarchical Decision Machine (HDM). The HDM is a
// robot control architecture developed to ensure autonomous and robust
// operation of a team of cooperating robots.
//
// HDM is currently included within the robofoot embedded software. It is used
// with the Groupe Robofoot EPM's multi-robot system, a system of autonomous
//  mobile  robots  with  distributed  sensing,cognition  and  control.  Groupe
Robofoot
// EPM's design is based on the RoboCup Middle-Size Robot League specifications
// (http://www.robocup.org).
//
// Copyright (C) 2004        Julien Beaudry
//                           Groupe Robofoot ÉPM (http://robofoot.polymtl.ca)
//
// This program can be used with Windows or Linux. It uses the following
// libraries:
//           Microb (http://www.robotique.ireq.ca/microb/)
//           Nlib (Sylvain Marleau)
//
// This software is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

/*
 * $Id: Action.C,v 1.2 2005/05/28 18:52:19 pmf Exp $
 *
 */

#include "Action.h"

Action::Action()
{
    creator = Creator::Instance();
    pointers_ok = 0;
    get_pointers();
};

void Action::get_pointers()
{
    pointers_ok = 1;
    braincontroller = creator->GetBrainController();
    perception = creator->GetPerception();
    ctrl = creator->GetController();
    motion = creator->GetMotionControl();
    data = creator->GetDataAcquisition();
    if(braincontroller == NULL)
    {
```

```
        cerr<<"Unable to obtain BrainController instance in HDM_node"<<endl;
        pointers_ok = 0;
    }
    if (perception == NULL)
    {
        cerr<<"Unable to obtain Perception instance in Action"<<endl;
        pointers_ok = 0;
    }
    if (ctrl == NULL)
    {
        cerr<<"Unable to obtain Controller instance in Action"<<endl;
        pointers_ok = 0;
    }
    if (motion == NULL)
    {
        cerr<<"Unable to obtain MotionControl instance in Action"<<endl;
        pointers_ok = 0;
    }
    if (data == NULL)
    {
        cerr<<"Unable to obtain Data instance in Action"<<endl;
        pointers_ok = 0;
    }
}

void Action::action(Action_parameters parameters)
{
}

void Action::update_info()
{
    if(!pointers_ok) get_pointers();
    else
    {
        cPosition MyPosition;
        cVitesse MySpeed;
        cPosition BallPosition;
        perception->GetPosition(MyPosition);
        perception->GetVitesse(MySpeed);
        perception->GetPositionObjet("BALLONORANGE", BallPosition);

        rbt_info.rbt_pos.x = MyPosition.X;
        rbt_info.rbt_pos.y = MyPosition.Y;
        rbt_info.rbt_pos.theta = MyPosition.Orientation;
        int id = braincontroller->GetRobotID();
        rbt_info.rbt_pos.id = id;
        if(id >= NB_PLAYERS/2) rbt_info.team = 1;
        else rbt_info.team = 0;
        int team = rbt_info.team;
        rbt_info.vtan = MySpeed.Vtan;
        rbt_info.omega = MySpeed.Omega;
        rbt_info.ball_pos.x = BallPosition.X;
        rbt_info.ball_pos.y = BallPosition.Y;
        rbt_info.ball_pos.theta = BallPosition.Orientation;
        rbt_info.ball_v = 0;
        rbt_info.ball_dir = rbt_info.ball_pos.theta;

        // TODO: obtenir positions des autres robots de perception
        int i;
```

```
        int crt = 0;
        for(i=0; i<(NB_PLAYERS/2); i++)
        {
            if(id!=i+4*team)
            {
                rbt_info.teammates[crt].x = OUTSIDE;
                rbt_info.teammates[crt].y = OUTSIDE;
                rbt_info.teammates[crt].theta = OUTSIDE;
                rbt_info.teammates[crt].id = i+4*team;
                crt++;
            }
        }
        for(i=0; i<NB_PLAYERS/2; i++)
        {
            rbt_info.opponents[i].x = OUTSIDE;
            rbt_info.opponents[i].y = OUTSIDE;
            rbt_info.opponents[i].theta = OUTSIDE;
            rbt_info.opponents[i].id = i+4-4*team;
        }
    }
}
```